FreeBSD® JOURNAL

July/August 2020

FreeBSD Friday Interview: Warner Losh

Building and Running an Open-Source Community

Easy Minecraft Server on FreeBSD

TrueNAS® M-SERIES
Powerfully Scalable Enterprise Storage

OPEN STORAGE FOR ENTERPRISE WORKLOADS

UTILIZES FLASH-OPTIMIZED ZFS TECHNOLOGY
IDEAL FOR LATENCY-SENSITIVE AND BUSINESS-CRITICAL VIRTUAL MACHINES AND PHYSICAL WORKLOADS.

- PERFORMANCE AND SCALE WITHOUT COMPROMISE
- INTELLIGENT STORAGE OPTIMIZATION
- SELF-HEALING DATA PROTECTION
- UNLIMITED SNAPSHOTS AND REPLICATION

Contact iXsystems to Learn More about what TrueNAS® can do for your business!

iXsystems.com/TrueNAS | (855) GREP-4-iX

iXsystems™

© 2020 iXsystems. TrueNAS is a registered trademark of iXsystems, Inc. All rights reserved. Intel, the Intel logo, Xeon, and Xeon Inside are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. All product names, logos, and brands are property of their respective owners.

FreeBSD® JOURNAL
Editorial Board

John Baldwin • FreeBSD Developer, Member of the FreeBSD Core Team and Chair of *FreeBSD Journal* Editorial Board.

Justin Gibbs • Founder of the FreeBSD Foundation, President of the FreeBSD Foundation, and a Software Engineer at Facebook.

Daichi Goto • Director at BSD Consulting Inc. (Tokyo).

Tom Jones • FreeBSD Developer, Internet Engineer and Researcher at the University of Aberdeen

Dru Lavigne • Author of *BSD Hacks* and *The Best of FreeBSD Basics.*

Michael W Lucas • Author of *Absolute FreeBSD.*

Ed Maste • Director of Project Development, FreeBSD Foundation.

Kirk McKusick • Treasurer of the FreeBSD Foundation Board, and lead author of *The Design and Implementation* book series.

George V. Neville-Neil • Director of the FreeBSD Foundation Board, and co-author of *The Design and Implementation of the FreeBSD Operating System.*

Philip Paeps • Secretary of the FreeBSD Foundation Board, FreeBSD Committer, and Independent Consultant.

Kristof Provost • Treasurer of the EuroBSDCon Foundation, FreeBSD Committer, and Independent Consultant.

Hiroki Sato • Director of the FreeBSD Foundation Board, Chair of Asia BSDCon, member of the FreeBSD Core Team, and Assistant Professor at Tokyo Institute of Technology.

Benedict Reuschling • Vice President of the FreeBSD Foundation Board, a FreeBSD Documentation Committer, and member of the FreeBSD Core Team.

Robert N. M. Watson • Director of the FreeBSD Foundation Board, Founder of the TrustedBSD Project, and University Senior Lecturer at the University of Cambridge.

Mariusz Zaborski • FreeBSD Developer, Manager at Fudo Security

S&W PUBLISHING LLC
PO BOX 408, BELFAST, MAINE 04915

Publisher • Walter Andrzejewski
walter@freebsdjournal.com

Editor-at-Large • James Maurer
jmaurer@freebsdjournal.com

Copy Editor • Annaliese Jakimides

Design & Production • Reuter & Associates

Advertising Sales • Walter Andrzejewski
walter@freebsdjournal.com
Call 888/290-9469

FreeBSD Journal (ISBN: 978-0-615-88479-0) is published 6 times a year (January/February, March/April, May/June, July/August, September/October, November/December).
Published by the FreeBSD Foundation,
3980 Broadway St. STE #103-107, Boulder, CO 80304
ph: 720/207-5142 • fax: 720/222-2350
email: info@freebsdfoundation.org
Copyright © 2020 by FreeBSD Foundation. All rights reserved. This magazine may not be reproduced in whole or in part without written permission from the publisher.

LETTER
from the Foundation

This year marks the 20th anniversary of the FreeBSD Foundation. At times, the day I signed the incorporation papers feels almost a lifetime ago: before I was married, before our two beautiful children came into this world, before teaching our teenagers to drive, before readying our eldest to head off to college, before moving parents to live close by, before serving as their caregiver. Intermixed in this span were new jobs and opportunities, celebrations with friends and families, excitement and disappointment, successes and failures, joy and sadness. All the things you'd expect from 20 years of life. And during that time, my ability to invest in FreeBSD, to focus on driving it forward, varied along with these life experiences. Again, as you'd expect across 20 years of living.

Twenty years ago, in anticipation of these types of changes, the FreeBSD Project's Core Team contemplated how to build a project that was stronger than the drive and passion of a self-selected group of its most prolific, early contributors. Two initiatives arose from this internal reflection: I incorporated the FreeBSD Foundation, and new project bylaws were enacted that transitioned the Project to an elected Core Team. Combined with the use of the BSD license, our 501(c)(3) nonprofit foundation and method of governance are powerful differentiators to other

LETTER from the Foundation

open-source projects. The Foundation provides a stable entity that is able to operate in the corporate world without being dominated by corporate interests. The bylaws encourage a renewing cycle of contributors and opportunities for leadership to any who come to our project. Together, these characteristics have given FreeBSD a life of its own, stronger and more enduring than that of any one individual. Twenty years on and many Core Team elections later, our success as a project has proven the power of this formula.

At milestones like this one, it is only natural to reflect on past accomplishments. For the FreeBSD Foundation, there have been many: protecting the Project's intellectual property; evangelizing FreeBSD at dozens of events every year; building out critical hardware infrastructure at co-location sites around the world; subsidizing new and veteran contributor travel to summits and conferences; improving release engineering, continuous integration, and FreeBSD's response to security vulnerabilities; and investing millions of dollars in underserved technical areas such as the early transition to SMP, the wireless stack, network congestion control research, building a BSD-licensed toolchain, the packaging system, and countless bug fixes and performance enhancements.

But our past represents just our beginning. Like the Project itself, the FreeBSD Foundation is well prepared to ensure that FreeBSD remains *the* reliable toolkit for innovation well into the future. Our strengthening partnerships that span from CPU and device vendors, to system integrators, security researchers, and solution providers, are growing the utility and usability of FreeBSD every day. Our outreach, internship, and educational initiatives are bringing new talent into the FreeBSD contributor base. Our funded development investments continue to grow in scope, ensuring—in partnership with the volunteer community—that we can tackle even the largest technical challenges. Each and every day, the FreeBSD Foundation explores new ways to progress our mission of bettering our world through the technology developed by the FreeBSD Project.

This first 20 years may feel like a lifetime to me, but human measure doesn't apply to corporations. I expect FreeBSD contributors reflecting back from the 40th or 60th anniversary of the FreeBSD Foundation will see these as "the formative years"—the years when the Foundation got a stable footing and then learned how to run. This journey is just getting started.

Justin T. Gibbs
Founder/President
FreeBSD Foundation

FreeBSD® JOURNAL
July/August 2020

12 **FreeBSD Friday**
The first of several introductory talks and 101 classes designed to get you started with FreeBSD. The program kicks off with Deb Goodkin's Intro to FreeBSD. *By Deb Goodkin*

17 **Interview: Warner Losh, Part 1**
Excerpts from a recent BSDNow conversation with Losh about Unix history plus his 2.11 BSD Restoration Project of which the Unix Heritage Society is part. *By Allan Jude and Benedict Reuschling*

25 **Building and Running an Open-Source Community**
The FreeBSD Project. *By Marshall Kirk McKusick*

36 **Easy Minecraft Server on FreeBSD**
A quick tutorial. *By Drew Gurkowski*

3 **Foundation Letter**
20th Anniversary of the FreeBSD Foundation. *By Justin T. Gibbs*

6 **New Faces of FreeBSD**
Some new and returning members of the Core Team introduce themselves and their goals. *By Dru Lavigne*

10 **We Get Letters**
Overclocking a Fusion Reactor. *By Michael W Lucas*

42 **Events Calendar**
By Anne Dickison

new faces of FreeBSD

BY DRU LAVIGNE

The election for the FreeBSD Project's Core Team took place earlier this year and results were announced on July 1. The FreeBSD Core Team constitutes the Project's Board of Directors, and as such is responsible for deciding the Project's overall goals and direction as well as managing specific areas of the FreeBSD Project landscape. Every two years, the Core Team is elected by the active developers in the Project. The 2020 election represents the 11th elected Core Team.

For this issue, we asked some new and returning members of the Core Team to introduce themselves and their goals.

Baptiste Daroussin, Clusteradm Liaison

I have used FreeBSD for around 20 years on my personal servers and as my main desktop/laptop. My contributions to the Project take place mostly during my free time. I am the author of poudriere, have worked on pkg unicode collation, replaced groff with mandoc, and refactored/modernized many parts of the ports tree such as the option framework, USES, and flavors. I have been a FreeBSD developer for 10 years (ports and src), a member of portmgr team for 9 years, and have done 2 shifst for portmgr in the past.

I decided to run again for Core to help the community on the nontechnical side. I want to help make sure that FreeBSD remains open for modernization and continuous improvements while keeping its strength, which is the quality of the engineering and the community nature of the Project.

Ed Maste, Git Transition WG and Security Team Liaison

I've used FreeBSD since 2003 or so. I was at a networking hardware company, and they began work on a new product built on FreeBSD. I got my start in the TCP/IP stack, which was undergoing fine-grained locking in the FreeBSD Project. Our product made some unusual demands of the network stack, so we often encountered issues that hadn't been seen before. I collaborated with FreeBSD developers to diagnose and report bugs and test fixes. This work led to becoming a committer in 2005. I remained with the same company for some time, leaving around 2012 to take on consulting roles, and since then have

been working with the FreeBSD Foundation to manage the Project's development efforts.

I previously served on two consecutive Core Teams, and then chose not to run in 2018 as a self-imposed term limit. After a term off, I felt ready to stand as a candidate again.

As the FreeBSD Project, I feel we can be rather resistant to change, but we also occasionally have pressure to make change without proper consideration or due care. Change is inevitable and is vital for us to remain relevant for new generations of computing users. I hope we can make FreeBSD even more interesting to students and others who are learning about and getting started with operating systems. I hope that they'll become FreeBSD users, developers, and future Core Team members.

There are two main technical projects that I plan to help move forward during my Core term: the transition to Git as our version control system and the full introduction of pkgbase. Core is not directly a technical body, but both of these projects require a significant amount of effort to be focused on coordination and communication with the community so that they can happen smoothly.

I believe it's vital for us to encourage new people with new ideas to help move the Project forward, and I am very happy to see a number of new, engaged committers on this Core Team, including two of my mentees in the Project.

One piece of advice I have for new Core members (and this goes for myself, too) is to figure out what's really important and learn how to respectfully decline tasks and topics that we don't have the capacity to handle. In previous Core Teams, we've had very long lists of tasks that were ostensibly being handled by the Core Team but, in reality, made very little progress. We need to focus the limited time and energy available from Core Team members on the most important tasks ahead of us, in order to make real progress.

George V. Neville-Neil, Foundation Liaison

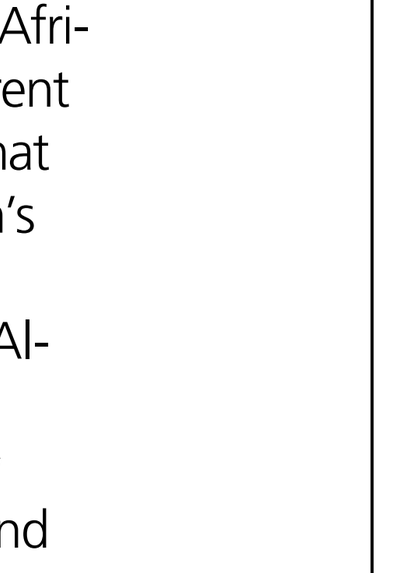
I first started working with FreeBSD code before I was a committer, porting the FreeBSD network stack to VxWorks in the late 1990s. I've been a FreeBSD committer since 2004 and have been on Core several times since then. My main areas of work are networking, from drivers up through all the protocols, and, more recently, DTrace.

I ran for core@ this term for two reasons: the first was that the global situation was especially fraught with Covid-19 shutting down much of the planet, and the second was that I felt it was time to reinvigorate the Project with some new direction. One of the key elements of being on Core—as I know from previous terms—is that we are flag wavers, which is how leadership proceeds in a democratically run, open-source project. One of the flags to wave is simply that we, as contributors, are all in this situation together. Many of us in technology discount the human aspects of our work, but in the current situation, it's quite obvious that keeping things together will be more challenging than it has been in previous years.

The second flag I wish to wave is one whereby we, as a project, refine and target our software and our message. The message is that FreeBSD is absolutely great at being a technology toolkit. FreeBSD should be picked up by pretty much anyone—from a single individual through start-ups to large companies—to produce systems and products that carry technology forward, producing high-quality, dependable, and performant systems. It's clear looking at our shared history that this has happened many times (NetApp, Isilon, Juniper, Netflix, and others are mar-

quee names in this area), and it should be where we continue to focus and set our sights as a project. It's my goal during this term to make sure that message is clear and consistent both inside and outside the Project.

Hiroki Sato, Code-of-Conduct Committee Liaison



I work as an assistant professor at the Tokyo Institute of Technology. In the Project, I am a member of the Release Engineering Team and Documentation Engineering Team, as well as a board member of the FreeBSD Foundation.

I began working on FreeBSD around 2000 and have been involved in various aspects of it since then, including the IPv6 stack, userland utility improvements, the documentation framework, and maintaining ports. I also have maintained one of the official mirror sites, which is for the Asia/Pacific region, and I host AsiaBSDCon, an international conference on BSD operating systems in the Asian region.

I decided to run for Core because I am interested in improving our community and I believe we should be aware of how geographically widespread our community has become. For much of the early history of the FreeBSD Project, most of our community members were in North America or Europe. The number of contributions from people in Australia, South America, Africa, the Middle East, Central Asia, or East Asia has been increasing, and with so many different regions involved, there have been increased opportunities for international collaboration. That said, there are also problems we should address. As a person in Asia, I can offer this region's viewpoint to the Project.

I would also like to help improve interactions between the Project and enterprise users. Although several such collaborations have been attempted, interviews with companies using FreeBSD indicate that the question of how to make a relationship with the Project mutually beneficial is still an open one. I believe we should discuss it and look beyond conferences and vendor summits.

I look forward to pursuing the goals of improving diversity and discussions concerning interactions with enterprise users. These are topics that require continuous discussion. And, we should have geographic diversity in our Core Team as well. While the number of people who run for Core in Asia is often small, I hope as an individual from Asia to provide ideas from a different perspective.

Kyle Evans, Portmgr Team Liaison

I'm relatively new, having only been with the Project for around four years. The majority of my work has been centered around userland and lualoader. FreeBSD is mostly a hobby for me—I've deployed a small fleet of FreeBSD/arm boards at work for various purposes, but they're pretty rock solid and don't require much work on FreeBSD. I'm generally drawn to the Project because it has interesting challenges to solve and, of course, I stay for the goat.

I wasn't initially planning to run for Core, but I was prodded toward the idea. Upon reflection, I decided that I wanted to run for Core and work on some of the social/communication issues I've noticed or even been bitten by in the past. My general goals are to improve

how we interact with each other and our contributors and to help the Project remain sustainable through committer retention and recruitment.

As the portmgr liaison for this term, my aim is to help portmgr improve as an organizational unit in all aspects. Some more specific goals are:

1. Reduce friction for non-portmgr proposed changes without lowering the quality of the ports tree.
2. Improve socialization of the direction for framework advancements.
3. Identify how portmgr works internally and determine if the dynamics are a match for how the team is organized.

Mark Johnston, Bugmeister and CI Team Liaison

I got into FreeBSD because of a student internship at a company that used it to build a networking appliance. At the time, I really wanted to get involved with open source and was interested in operating systems, and, fortunately, I managed to get a job on the fairly small OS team at this company, which happened to contain a couple of FreeBSD committers who helped mentor me into the Project. Since then, I've worked for several appliance vendors, and I now do freelance FreeBSD development. Many of my contributions have been in the form of bug fixes for the kernel—for some perverse reason, I enjoy trying to track down obscure concurrency bugs. I've also done some work on our DTrace port and on the VM subsystem.

I ran for Core because I wanted to get a better handle on the communication challenges of the Project. FreeBSD has a large, distributed developer base, together with many downstreams that vary in their level of interaction with the FreeBSD Project. At the same time, FreeBSD is also a much smaller and more cohesive project than certain other well-known, open-source operating systems, and as a consequence, it's often possible to make quick progress on problems by getting the "right" people involved. This doesn't happen as often as it could, especially with large downstream vendors, and I would like to better understand the needs of our downstreams and developers and try to identify the gaps in our development process. Right now, I think it is still overly difficult for non-committers to get patches submitted, reviewed, and committed.

With the transition to Git, I expect to see an even larger volume of patch submissions from contributors, and the distinction between "contributors" and "committers" will become blurrier. As a member of core@, I would work on policies that help define and streamline the workflow for contributors to submit patches and for the committers who test and review contributed patches.

DRU LAVIGNE is the author of *BSD Hacks* and *The Best of FreeBSD Basics*.

WeGet letters

by Michael W Lucas

Overclocking a Fusion Reactor

I don't know how many of you have written to me whining about performance because I bin those letters as soon as I realize what they're about. Sometimes that takes a few minutes, because so few are capable of composing a letter that complies even minimally with English language standards, distressingly low as they are. The *FreeBSD Journal* is a classy establishment. We have not, nor will we at any time in the future, accept letters via text message, as most carriers have banned the emojis most suitable for response as unspeakably obscene.

I do, however, feel obliged to throw a few of your queries into the blender, and I see some readers want to know how to optimize performance.

I have suggestions. Many suggestions.

They all come down to: don't.

The natural impulse to compare your server against others, to whup your neighbor, is exactly that: natural. It's like flipping rocks to find tasty grubs. Sleeping in trees coveting the upper-class caves inhabited by the snooty grizzly bears. Perishing within 20 years of What Is Vitamin C Anyway? We invented civilization to escape dysentery, cleaning ourselves with mud—and benchmarking.

This is so natural, it's even in the most sacred of geek cultural touchstones: Star Trek. In Deep Space Nine, O'Brien is constantly fiddling with the Cardassian space station to get its performance to a level he finds acceptable. Not the station commander's standards. Not the station's inhabitants' desires. A level he finds acceptable. Truth is, the space station works just fine. The space station resists his tampering because its software knows how the systems are *supposed* to behave, and his obsessive tinkering threatens its integrity. Someone combining my Trek gene with an Obsessive, Irrelevant, Detail-Measurement gene has certainly counted how many times O'Brien almost killed hundreds of people by insisting on overclocking an alien fusion reactor. (The United Federation of Hold My Beer simultaneously explains all Star Trek plot holes *and* all Star Trek plots. Look it up.)

Maybe it's your job to nurture the server and you want to know that you're doing your job well. The machine doesn't get a vote on how you're doing your job, though. IPC with your organizational parent process instead. It's your manager's job to tell you how you're doing.

But no. Some of you will insist you can eke a few extra percent of performance out of the system. The fact that everyone might wind up sucking vacuum is a risk you're willing for them to take.

The only performance that matters is what the user experiences.

So, stir your lazy carcass and communicate with the users.

Preferably by voice. Writing is a horrid communications mechanism at the best of times.

If you can manage it, communicate in person. Barring plagues, of course. *Two-way* communication. Converse with your users.

This means listening. You can't use the time people are babbling ignorance at you to figure out what you're going to say next. You need to process their babble and extract meaning from it. Preferably, the meaning they intended. Present your extracted meaning to the user and

request verification. I guarantee that your users' suffering has no bearing on how many disk writes per second the database groans out. If by some chance such things are relevant, I guarantee you then an extra one or two percent performance is not going to ease their woes.

You won't enjoy this process. Sysadmin Rule #2 is very clear: "People were a mistake."

But the system exists to serve the users.

Many of the things they disclose will appear to be not your problem. If they're trying to run your fancy client-side JavaScript application on a prototype Atom processor from the previous millennium, it will be slow. Is the problem that their desktop should have been sent to the Giant Heap of Eternal Toxicity a decade ago? Is your application wrong for this organization? Were both desktop and application approved by a computer-illiterate whose primary qualification for the Purchasing gig is an ability to keep the Chair of the Board of Directors' most troublesome child from constantly calling Mommy with their brand-new Distressing Drama of the Day?

Will benchmarking help you with the issue? No. Discover the real problem. Solve it.

If the lure of the natural urge to benchmark proves too strong, though, sublimate it in useful directions.

Sure, set up monitoring to be sure that your application responds within reasonable time limits. That's more of an operations issue. Graph the results of that monitoring so that you can see how performance changes over time. When someone says, "the application seems slow after the upgrade," that graph will either let you declare that you're already aware of the issue or will justify your scorn.

Beyond that, target a user. You know the one. The Problem User.

Benchmark *them*.

Fire up your favorite packet sniffer. Watch the interaction between the server and Problem User's machine. Capture packets. Analyze them. See what happens in the real world. Take notes.

Dtrace, ktrace, truss. They're all your friends. Watch the server do its job. Where does your application spend its time? Take more notes and samples.

Perhaps you can find a real problem that affects Problem User. If not, you'll learn an unwholesome amount about your environment and your applications.

Eventually, you'll get called into a meeting with your organizational parent process. Maybe it's an annual review. Perhaps Problem User has finally made enough noise that Management has no choice but to stir themselves enough to make interested noises. When they ask what you've been doing, you have a stack of data to show them. Perhaps it's not useful data, but that's okay. You've delved deep into the innards of your environment. You can demonstrate that you have spent time investigating Problem User's issues. Packet traces and Dtrace results, heavily annotated with your comments and arrows and swooping lines connecting different sections, never fail to impress.

Yes, benchmarking is a natural human thing. And there's nothing more *natural* than a business.

You, personally, are constantly benchmarked. Make your benchmarks solid.

It's almost as much fun as overclocking a fusion reactor.

Have a question for Michael?
Send it to letters@freebsdjournal.org

MICHAEL W LUCAS (https://mwl.io)'s newest books are *Sudo Mastery, 2nd Edition* and *Terrapin Sky Tango*.

FreeBSD Friday

BY DEB GOODKIN

The first of several introductory talks and 101 classes designed to get you started with FreeBSD. The program kicks off with Deb Goodkin's Intro to FreeBSD.

I'm Deb Goodkin, Executive Director of the FreeBSD Foundation. Welcome to our FreeBSD Friday presentation, the first of many introductory topics that we'll cover over the next few months.

Many of you may be new to FreeBSD, but I'm also making the assumption that we have people online from our community, and so this will be a very basic introductory-level, informational session on FreeBSD—and probably a refresher for some.

As I myself am also constantly learning more about FreeBSD, I will be watching these talks and learning alongside you. So, thank you for joining in this adventure with me.

Today's presentation is a very basic introduction to FreeBSD. The next two FreeBSD Fridays will be about getting started with FreeBSD with sessions Installfest, Parts 1 and 2, hosted by Roller Angel. The idea is that you can watch him for the hour, which will be recorded. Later, when you're ready to install FreeBSD on your system, you can access the sessions and follow along at your own pace.

Following Roller Angel's sessions will be Dr. Marshall Kirk McKusick (History of the BSD Fast Filesystem). Most of you know him as Kirk McKusick, who created the Fast Filesystem. It'll be a recorded talk, but Kirk will be available for a Q&A at the end. And then the last session we have planned will be An Introduction to ZFS by Dan Langille. You possibly know Dan not only from the community but as the person who runs BSDCan.

I encourage you to keep an eye on the Foundation's webpage for more of these FreeBSD 101-level talks, which will all be geared toward introducing you to various areas of FreeBSD.

I already introduced myself but here's a little more background. I joined the FreeBSD Foundation in 2005. Before that, I was a developer in the storage world and worked mostly on disk-drives—starting out as a firmware engineer, then moving into different areas of research and development and then to marketing and technical sales. As the Foundation's purpose is to support the FreeBSD operating system and community, it's really important for me to understand it. I have personally found studying it really interesting and enjoy learning about it.

The Foundation's purpose is to support the FreeBSD operating system and community.

The goal today is to give you a very basic overview of FreeBSD, why people use it, how the Project works, and why you should use it and contribute to FreeBSD. Our goal is to get information out there, to educate people, and to get more people using and contributing to FreeBSD.

Let me say that even now, in 2020, some people are still not clear that FreeBSD is not a Linux distribution. It can be very confusing when someone asks about your favorite Linux distro and they

include FreeBSD as one of the choices. So, we just keep making sure people understand that it's not Linux. It's not a Linux distribution. And yet, I know the reason they include FreeBSD is because FreeBSD is so popular. The question should have been what's your favorite operating system?

FreeBSD is comprised of three different components, the first of which is the actual operating system and that's what you're going to load on your computers with Roller Angel's guidance. The second component is the community—people who are watching this livestream, those who are interested in learning about FreeBSD and would like to start using it, trying it out, playing around with it, and contributing. The third component is the Foundation, which I run. It is actually a separate organization and I'll talk about what we do before we conclude today.

So, what is FreeBSD? It is one of the oldest, largest, most successful open-source projects in the world. It's a complete operating system, and we support over 33,000 third-party software packages. We have a community of highly technical and committed contributors and over a thousand contributors to the Project. FreeBSD runs on all of the major platforms out there, and we continue to make sure that FreeBSD runs on newer architectures, too, as well as on most of the main cloud providers. There truly are over tens of millions of deployed systems out there.

It is one of the oldest, largest, most successful open-source projects in the world.

A little history now: FreeBSD descended from BSD, which came about from the original AT&T UNIX. And both FreeBSD and NetBSD branched out from BSD, and then OpenBSD branched out from NetBSD. When you have a chance, take a look at the timeline on the Foundation website as it's really interesting. UNIX was developed in 1969. It came out of the labs of AT&T, and then Berkeley actually started improving and modifying it in 1974. The first unencumbered operating system, 386BSD, came out in 1992. And it branched out from there. We celebrate BSD's birthday on June 19 because that's when the name was chosen—Berkeley Software Distribution. We've enjoyed 27 years of innovation and growth.

One thing I want to mention at this juncture is that there was a relevant lawsuit in 1992. I bring it up because it lasted for around two years and it really limited the development and adoption of BSD during that time. Most of the developers who were working on it got focused on that lawsuit. And it really slowed things down, and as some of you may know, that's actually the same time Linux came around. Sometimes, people will ask why did Linux take off? And one response is that the lawsuit was actually the reason or certainly one of the reasons.

When UNIX was developed in 1969, most major computer companies used it and created their own versions of UNIX. So, you had IBM and HP and Sun. They were all proprietary versions.

Most of the open-source versions that came out of UNIX were from Berkeley UNIX, and FreeBSD has lineage from Berkeley UNIX, which came from the original UNIX. Linux, however, came about all on its own, and it came out with its own open-source version of UNIX. If you'd like to find out more about the history, there are many interesting online resources.

A lot of companies use FreeBSD. Most likely, you're using FreeBSD. You may know that Apple originally used components of BSD and that there is a lot of FreeBSD that was or is in those operating systems. If you're watching a Netflix movie and you're streaming it, it's running on FreeBSD-based servers. And if you're using a PlayStation, that's FreeBSD-based. People may not realize it, but there are a lot of products and companies that base their products on FreeBSD. We're just trying to raise awareness of that.

So why use FreeBSD? Well, there are different reasons if you're a user, if you're developing a product, or if you're doing research. And there's the broad community of users. Why do

people join open-source products? Why do they contribute? A lot of times, it's because of the technology they're interested in. But when you look at the thousands of open-source projects out there, there are a lot to choose from, and many times, the choice ends up being due to the community.

There's also a high regard for our excellent documentation. And that's not only because of access but also the ability to find most of it in one location, plus all the manual pages, and you can have the handbook installed, too. It's up to you. Remember, it's a small base system. You get what you need, and then you add the software packages that you want to do what you need to do or want to do. Those packages are up-to-date and you get them from one location. If I want to run Firefox, I don't have to go to the Firefox site. It's all on our own infrastructure.

As I mentioned earlier, there are a lot of CPU architectures that are supported—and then there's the BSD license. If you're just an end-user, if you're putting it on your desktop like I do, maybe you don't care so much about the BSD license, but a lot of developers care about it as do a lot of commercial users. And it's stable and secure. It's reliable in its performance.

Let's move along to how the FreeBSD Project works. The FreeBSD Project is independent of the FreeBSD Foundation. It is led by a Core Team. You get your commit bit through a mentorship process. You have one community that supports FreeBSD and it's made up of individuals and functional teams that put the operating system together and support it. It is a collaborative development model.

You have one community that supports FreeBSD and it's made up of individuals and functional teams that put the operating system together and support it.

We have two high-level functions or organizations. The FreeBSD Foundation is an organization. The FreeBSD Project is the volunteer arm. Under the Project, you have a Core Team and under them, you have many different functional teams like Security Team, Document Team and so on. You have all of these different teams that work together, and they are the ones who produce the product. There are many different teams of people who support FreeBSD. This gives people teams to be part of and allows them to contribute in different ways.

The Core Team is elected every two years. They're self-nominated and then voted on by active committers—and there are currently around 400 committers. They are all volunteers, and they typically meet every two weeks or at least once a month.

If there are any conflicts within the community, Core may step in to help. They also help provide strategic direction. But they don't step in and say, this is what we're going to do. No, in our project people want to work on the things they want to work on. But Core can help encourage people to work in certain areas that we know will help strengthen FreeBSD. We don't have a benevolent dictator. We don't have a 1% who oversee the Project and who make all the decisions. It's a community.

In terms of age range, we do have a lot of younger people who join the Project. Most are between 30 and 40. But we also have the older, more experienced, more knowledgeable people. And one nice thing is that a lot of the older people have been with the Project since the beginning.

Some of these people were with BSD when BSD was at Berkeley, which is really cool because they bring that very particular experience and knowledge with them. Younger people

can go to the more experienced people and learn from them. The people who've been with the Project forever are very approachable. And so, you can go to a conference, meet them in person, and they can tell you why decisions were made, why we changed things up, how we did it when we were at Berkeley, and why it still works now. And so, it's sharing and handing over that knowledge, that experience, which is really what helps make this project vibrant and sustainable. And I should add that there are many countries represented, which makes for a diverse, global community.

In terms of the particular community philosophy, the Principle of Least Astonishment (POLA) means that changes are only made to improve, to add features, and to make FreeBSD better. It's not a community that makes changes for the sake of making a change. And because of that philosophy, the changes don't typically break things. And because it works so well, that philosophy is followed across the Project.

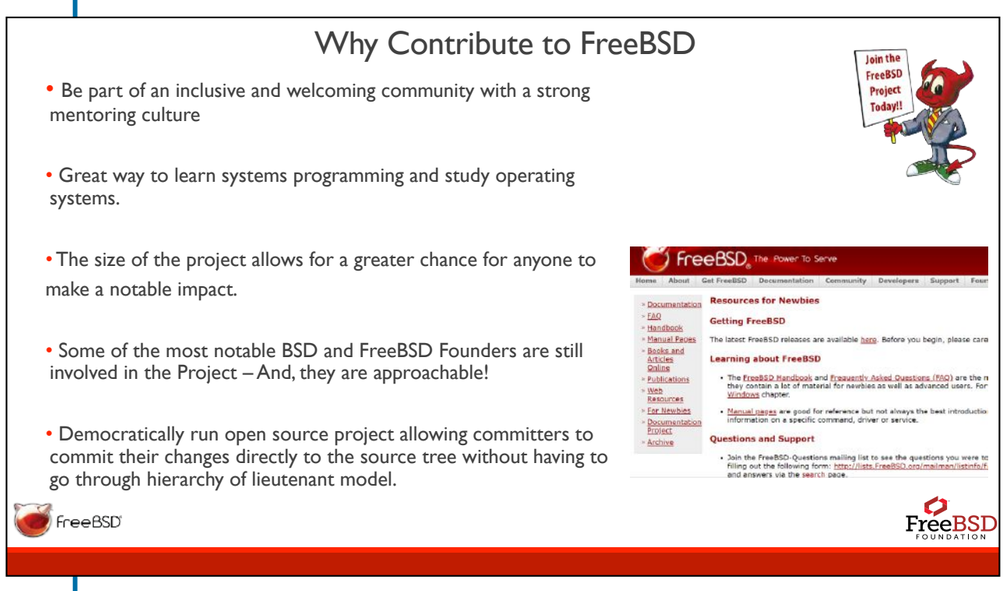
Regarding releases, we have two major types of releases—the major release and the point release. The major releases, for example, are when 12.0 came out or when 13.0 will come out next year. So, major releases happen every couple of years. The point—the in-between—releases fall more in the category of something like security fixes that are added. And those can happen about every nine months.

How can you contribute to FreeBSD? Let me count the ways!

And we have two branches of the source code. We have a Current and Stable. If you make a change and you commit it, that goes into the Current branch. And Stable is the version where those changes that went into Current appear after they go through testing. Only then may they be moved over to Stable.

How can you contribute to FreeBSD? Let me count the ways! Usually when you think about a software project, you think, well, I have to contribute by writing code. And, yes, that's the most obvious way. But there are so many other areas that are just as important—for example, documentation. When I was talking about how we're known for our excellent documentation, I was referring to doc committers. People who can commit documentation are viewed very highly in the community as are those who maintain the ports—those 33,000 or more software packages that are out there.

And advocacy, which is what I'm doing right now, can be fun. There are people who have their own YouTube channels, people who are doing their own videos on FreeBSD, creating how-to guides, writing up blogposts about a special feature in FreeBSD and how they got it working. There are so many different ways that you can contribute!


Why Contribute to FreeBSD
• Be part of an inclusive and welcoming community with a strong mentoring culture
• Great way to learn systems programming and study operating systems.
• The size of the project allows for a greater chance for anyone to make a notable impact.
• Some of the most notable BSD and FreeBSD Founders are still involved in the Project – And, they are approachable!
• Democratically run open source project allowing committers to commit their changes directly to the source tree without having to go through hierarchy of lieutenant model.
Join the FreeBSD Project Today!!
FreeBSD
FreeBSD Foundation

And it really is easy to get started contributing. My suggestion—even if you're a software engineer—would be to start by contributing documentation. By doing that, you're learning about the operating system. You might update a man page, and then you learn about how that command works. Or you may translate into your native language. By doing that, you're learning as you go. We can always use people to improve documentation. We can always use ports maintainers, or if you have a software package that you love and isn't supported, you could bring it into the fold and support that.

My other suggestion is to go through the PR (Problem Report) list and fix some bugs. We can always use people to do that.

But let's move on to a little on why companies use FreeBSD.

When I go to a company and talk to them about the use of FreeBSD, they already know about FreeBSD. They usually know the history and they feel comfortable using it because of that. And companies tend to like the POLA philosophy and knowing that when they install FreeBSD and they have it on their systems or in their product that it's going to be really reliable and stable. And when we come out with new releases, they understand that everything will work. Companies use it for its high performance and the tools, the mature release model and the excellent documentation.

One of the most frequent reasons companies tell me they use FreeBSD is the BSD license. It's a permissive license, and what that means is that they can put their own IP, their own secret sauce, their own stuff, into that code. They don't have to give it back. However, we find that most of the time they do give back most of the code upstream (except for their IP), which is really beneficial to them as well as to us.

We also hear from you. We hear from the community.

The other most frequently cited reason for using FreeBSD is the ZFS support.

By the way, I highly recommend a book called The Design and Implementation of the FreeBSD Operating System [Marshall Kirk McKusick and George V. Neville-Neill]. I recommend it because it is a great resource for learning about the internals of the kernel.

I'm going to conclude by talking a little bit about the FreeBSD Foundation. We are classified as a 501(c)(3), which is a U.S.-based tax code. And what that means is we're a nonprofit public charity and really work for the public good. And so our whole focus is supporting the FreeBSD Project and community. We are funded 100% by donations from individuals and commercial users. We will get individual donations from $5 to $50,000, and corporate donors typically will donate between $10,000 and a million dollars. That's how we keep going, and we use that funding to do the work that we do to support FreeBSD.

We are separate from the Project and we don't oversee the Project. We don't tell the Project what to do. What we try to do is assess what the Project needs—especially the most critical needs. And we try to step in to help in those most critical areas. To do that, we meet with companies to understand their uses and their pain points and their needs. We ask what they need to start using FreeBSD or to continue using FreeBSD.

We also hear from you. We hear from the community. We keep our eyes on mailing lists and social media and we keep an eye on the questions that come in. We listen to people who say they're having problems with things and we try to help.

We fund projects internally and we fund projects externally and we do a lot of advocacy. Since we're not going to conferences right now—which we typically do—we have started this FreeBSD Friday Series to introduce people to various areas of FreeBSD and to help us all learn from one another.

Let's say you want to learn about operating systems. FreeBSD's smaller code base is a great way to learn about operating systems. For example, our kernel is just over 5 million lines of code compared to Linux, which is about 26 million lines of code. And this is just the kernel. So, if you want to actually look at code, if you want to look at the source code, there's a lot less to look at in FreeBSD, which makes it a great way to learn.

Why chose our project? Part of the reason is that we're inclusive and welcoming and that we have a strong mentoring culture. Because we're a smaller project, you can make a more notable impact.

AN • INTERVIEW • WITH
Warner Losh

by Allan Jude and Benedict Reuschling

BSD Now Episode 362, 2.11 BSD Restoration. Recorded for August 5, 2020.

Benedict Reuschling and Alan Jude speak with Warner Losh about Unix history and its interesting tidbits. They discuss his 2.11 BSD Restoration Project, of which the Unix Heritage Society is part. And what's that thing he wrote called Death Match? We hope you'll enjoy it all! We present Part 1 in this issue and Part 2 in the next.

REUSCHLING: Welcome, Warner. First, for the people who don't know you as yet, would you tell us a little bit about yourself and how you got involved with Unix and BSD?

WARNER LOSH: You bet! I have been involved with Unix since my college days. I started on a VAX-11/750 running 4.2 BSD back in the early 1980s—it was 1984, I think. And I've been a BSD user for a long time, and a FreeBSD user since the company I was working with ported our software to FreeBSD 1.0 GAMMA as Jordan Hubbard also happened to be a client of ours.

I've done a lot with FreeBSD over the years. I was a security officer for a while and have served on several Core Teams. I've done PC card, CardBus, devmatch, and some general storage things lately having to do primarily with optimizing flash performance for my company's video streaming servers that you might be familiar with. I work for Netflix, and they graciously let me optimize the FreeBSD storage system.

REUSCHLING: So, we can blame you for a lot of good things?

LOSH: Yes, but if your set-top box doesn't play immediately, don't call me. We have a lot of good people for that who are better at troubleshooting those sorts of things than I am.

JUDE: You're such a BSD history buff! I look forward to talking with you about the project you're working on, which is apparently older BSD than the first BSD you started with.

LOSH: It was actually released after, even though it has a lower number.

JUDE: Right, it will be interesting to hear about that. Recently, you spoke at FOSDEM about the early history of Unix, and then later gave a related talk at BSDCan. Do you think Unix history matters even more today?

LOSH: I think Unix history matters for several reasons. Unix was an innovative system in its day, and it drove a lot of very interesting innovations, some that people seem to forget about and then rediscover. And by knowing and understanding history, you can understand some of the early issues and problems with different approaches, and you can also better understand the larger context of how we got to where we are today and why we do certain things and don't

do others. And history adds an understanding of why there's a philosophy of Unix, and yet there are some things that don't quite fit the philosophy, like, for example, sockets and how that came about and how it evolved and how it how it turns out people actually tried to make networking stuff fit just a normal file descriptor and found problems with that. By understanding all of those happenings, you can understand why we have some of the things we have today.

Also, by looking at the early Unix—if you're a kernel hacker, looking through early Unix code—it's a lot easier to approach, a lot easier to read, a lot easier to understand what's going on. And in a lot of ways, the core algorithms that were used at that time are still present in the kernel today and modernized mostly for core preemption for faster CPUs and different technology. It'll help you understand what's going on, which will give you a jump on understanding the code if you go to read a modern BSD kernel.

JUDE: It's interesting how many of the concepts we have today came from Unix and basically haven't changed. Like pipes. Basically, every operating system has pipes now, and it wouldn't make sense not to have pipes. If it wasn't for Unix, how would we have gotten any of those things, even those that aren't really related to Unix anymore? So many core concepts came from Unix.

LOSH: Exactly. Pipes were this thing where people thought well, yeah, we'll try this, it might be interesting to have a bunch of small tools to string together on these old PDP-11s that didn't have much address space. The tools had to be simple because that's all you had space for, and that notion has carried over since then into a number of other operating systems. It's quite interesting.

REUSCHLING: Oh, yes, especially if you have a history background and know where it originally came from and how it was implemented. I teach a lot of students every year and the Unix history sections are unfortunately often where I lose some of the students, not completely, but some fall asleep. Do you know a good way to teach students Unix history without it being boring?

LOSH: I don't know if I have a good way of teaching students without being boring. I'm not a teacher. I don't have students in front of me so I don't know if these things will work. But one of the things I do when I'm just talking to people at conferences or over a beer, or whatever, is relate how the different historical developments fit together. If you read the history of the quarter century of Unix, you'll get a good sense of what happened when, but you won't know something like the people who were developing it had a friend at Bell Labs who was feeding them new tapes—kind of on the sly—so that they could keep things up-to-date. There are personal details that make it all very interesting, and those aren't always public.

Some groups like knowing that Dennis Ritchie spent some time at Berkeley, and that Unix got so established at Berkeley because Dennis and Ken spent time there. I think I may have just misspoken, it may have been Ken that spent a semester on sabbatical there!

Knowing that Unix is more than just the technology, that Unix is also the people and the connections between the people and how those played into how Unix spread and was adopted is extremely interesting. If you can help people relate to how the different pieces fit together, more than just the recitation of facts, that seems to work better. I know it works better in my talks with adults. I imagine it would work well with students.

JUDE: I think some of the most interesting parts of Kirk McKusick's talks are the personal stories. There are just so many interesting bits that make it interesting. And I think another thing is that to really appreciate Unix history, you have to first understand some of Unix so that you can then appreciate the elegance of the way it was developed and also how it so easily could have gone very differently if there were other people involved or if Person A hadn't met Person B or even if they both hadn't been at the same place at the same time.

LOSH: Exactly. There are a lot of person-to-person situations, and there were bakeoffs. One of the reasons 4.1 turned into 4.2 BSD was that there was a bakeoff between BSD Unix and VMS. And they did a bunch of dumps of performance data to see different workloads so that 4.2 BSD could be improved. A lot of things that they did in these bakeoffs was to spur Unix development, to make it faster, better, more robust. In the end, Unix wound up winning because it was faster, better, and more robust for the things they needed.

JUDE: To Benedict's point about teaching, a lot of it is about making it matter to people. The other part is looking at just how many bits of history we've already lost, whether that's when somebody had to rewrite a program based on only the disassembled original binary or that a tape has been lost forever and so on. It makes us think about the code we're writing now and make sure that we preserve it in a way so that in 30 or 40 years from now when somebody wonders how we did that, not only will they find the source code, but also the things we don't always think about, which are the tools around it.

For instance, to be able to build Free BSD 4, you kind of need a Free BSD 4 system as you're not necessarily going to be able to just compile it on whatever computer you have 30 years from now.

LOSH: That's true.

JUDE: We might have to think about preserving more than just the source code history from GitHub or whatever.

LOSH: Yes, having binary artifacts is important so that you can go back and recreate things. It's quite useful to be able to run the original binary even if you have source. Some of the early Unix source is written in Dennis Ritchie's style, which is pretty fancy, and a lot of modern compilers just do not like it at all. If you spend a lot of time and effort, you can find all the right switches to make it swallow that code, but there have been subtle language changes, and not everything works if you do that. Having the original binaries and emulators that can run those binaries is quite important for recreating history as well, should you want to do that.

JUDE: Yes, and speaking of recreating history, tell us about your project to recreate BSD 2.11 I guess patch-0.

LOSH: You bet. For those who don't already know, you know it as 4.2BSD that was in the fourth series from Berkeley. They did a 1BSD, a 2BSD, and a 3BSD. 1BSD and 2BSD were for the PDP-11; 3BSD and 4BSD were for the VAX.

Berkeley tried very hard to run away from the PDP-11 versions, but they were so popular that they wound up doing several releases. And we have those releases. We have copies of the master tapes, but it went from 2 to 2.8 for some reason that I don't know. I think it was

because it was the 80th tape. There's a BSD2.78 which has notes around it saying there have been about 70 tapes released. So maybe that's why, maybe not. But there's a 2.8, a 2.9, a 2.9.1, a 2.10, 2.10.1 and we have all of those releases. And 2.11 went out through USENIX, not through Berkeley. USENIX managed all the tapes, and there was actually some data corruption on the early tapes. Actually, the corruption was not on the tapes but on the system that made the tapes, so there were blocks of nulls where they should have had the source code to the games.

Nobody preserved the early tapes. USENIX said, oh, there's some corruption, let's come up with a new thing. And so, they remastered the tapes and the earliest one that survives is at patch level-195, which is 18 months or so after the original release. So, all of the changes in that period were preserved.

Then we thought, oh, you can just run it backwards, you know, un-apply the patches—as we're all used to modern patches, you just do path minus r, everything's cool, right? Well these patches were posted to Usenet in a number of different formats. And when they were posted, some were in a style like—run this shell script, or extract this tar ball. And when you did that, the shell script would remove files, or the tar ball would overwrite files. We don't have any extra context to go back to. We can't run the sausage mill backwards to make pigs like you can with most modern systems.

Every so often I would ask people, hey, does anybody have an original tape? And people would just say no. As a result, I just kind of let it sit there.

Then the coronavirus hit, and it caused me some anxiety and sleepless nights. You can either stay up and worry about things or you can distract yourself. So, this started as a distraction. I thought, well, let's take a look at the patches. What information was destroyed? It turns out that a lot of files were brought in to replace stuff that was in the previous version—so I could just grab the previous version. And that got me a certain distance, but there were about a dozen files that weren't quite right, didn't compile, didn't work.

Thank you!

The FreesBSD Foundation would like to acknowledge the following companies for their continued support of the Project. Because of generous donations such as these we are able to continue moving the Project forward.

FreeBSD™ FOUNDATION

Are you a fan of FreeBSD? Help us give back to the Project and donate today! **freebsdfoundation.org/donate/**

Please check out the full list of generous community investors at freebsdfoundation.org/donors/

Uranium

Iridium

Platinum

Gold

Silver

I was then able to assume, well, they got this from 4.3BSD. And for about half of the files, I could get material from 4.3BSD because there was an updated 4.3BSD and the changes were bigger. So, okay, now I've got that.

But one of the things that they changed from 2.10 to 2.11 was the archive format. There was an old PDP-11-specific archive format, plus the new, modern archive format that they decided to switch to because it would make things easier. They could do more things on the VAX, and cross compile and take it to the PDP-11 because the PDP-11s were really slow machines in comparison to the VAX.

So, they did that, and I was able to pull in that code and merge it with the BSD2.10 and the BSD4.3 code. And eventually, I was able to have a string of patches all the way back to the beginning that was consistent—and by that, I mean that all the information that's in the patches, all of the reconstructions I did, didn't conflict with anything. But that's kind of a low bar—to not conflict with anything. You can have an empty file not conflict and still it can be totally wrong because there was content to that file.

And so, I started thinking, well, you know that's an interesting outcome so far, but let's see if I can actually build the system. And I got into bootstrapping the system and ran into all kinds of issues around that. I could talk for an hour and a half about them, but a lot of the issues are kind of boring.

The biggest issue was that they changed the assembler enough so that the system I started with couldn't assemble the old system at all. I had to figure out how to use the version-7 assembler to assemble BSD binaries because the system call formats had changed between version 7 and BSD. I couldn't just run the version 7 binaries; no, that would be too easy.

I had to figure out how to do that and use the version 7 compiler to compile the BSD system calls to link that all together using a PDP-11 simulator—I wanted to do that without having an actual system. So, the first step of the bootstrap was to build that and then use the oldest 2.11 system we have—the patch level-195 system—to build a couple of things. And then I had the bare minimum tool chain I needed to generate the old binary format because one of the things that changed between patch level-0 and 195 was that patch level-0 had all the old limitations.

There were eight-character identifiers and a bunch of other limitations that they wanted to get away from. They changed the binary format so they could have longer symbol names. And that presented a problem. I couldn't use the new compiler to build binaries for the old system because then you're mixing binary format types.

Once I had the basic tool chain bootstrap, I was able to use that to then bootstrap everything else. Last night I was able to create a boot tape for the original system for the first time, and I'm testing that out. And when I'm sure that's good, I'll be able to release it.

But in doing this, I realized, oh, this piece is missing or that piece is missing, or that compiles but doesn't work. And I've gone through and debugged all of those things in the six months or so since I've had sleepless nights. I'm getting very close to being able to announce, hey, here are the tapes, you can go try for yourself—these are the setups you need to use with the simulators, and here's the source code.

Once I have that, I want to make each individual change publicly available on GitHub someplace so that people can go through and look at the history and understand the history of the system. There were a lot of developments and a lot of innovations that happened during the patches that are really hard to access now. Getting at each individual piece is difficult, because they are in so many different formats, and there's not a convenient way you can browse through history. When looking at history with GitHub's history browser or the

GIT command's history browser, I can find where things happened, and it becomes a much more accessible history. Then it's not history that's locked up. I liken it to having a place people who are interested in this can come and look at it and be able to then understand what's going on and see the progression. And they can see the interesting tricks—the PDP-11 has a very small address space. It's a 16-bit machine. It has a 64kilobit address space which through MMU Magic doubles, you have 64kilobits for code and 64 bits for data. And then there's some overlays you can do to swap code in and out, kind of a precursor to dynamic paging that you have in 32-bit systems.

Looking at those techniques, you can see ways to make code fit in smaller spaces and ways to simplify the code that doesn't sacrifice functionality and allows it to fit in the space constraints available.

So that's why I got started on this project. Plus, people told me I couldn't do it, it wasn't possible, there's no way I'd succeed, which also is a motivator.

That's the BSD2.11 project, and if you're one of my Twitter followers (I'm bsdimp on Twitter), you can see the little teeny, tiny, incremental steps of progress, or if you read my blog, you can read the bigger chunks of progress. I don't know when this discussion is going to be airing or read, but in the coming weeks, there will be pointers to the posted artifacts. And when the GitHub stuff is done, that'll come up.

JUDE: When that's done, you'll have each of the patches as a GIT commit?

LOSH: Yes. Each patch is a GIT commit and was sent to a Usenet news group, and each of the patches has a header on it that will be the commit message that describes what the patch does plus any interesting commands you need to run. Also, as part of this, I'll have a series of scripts to recreate each of the patches.

To go backwards, it's a lot of manual work so I automated it. You know, I got back to patch-50 and I found a problem in patch-175. It stuck me that was 150 patches I'd have to redo by hand. Well that got old in a hurry and so I automated it and I automated the bootstrap process.

Now, with just a couple commands, I can go from historic artifacts to a tape that has the original—my reconstruction of the original 2.11BSD—on it. And that allows me to go almost to the end and find, oh, I'm missing the boot loader for this type of hardware that I know was in the original release—let me go find it. But let's just regenerate it all so I don't have to sit there and type. I can regenerate it and go get some lunch or grab a beer or whatever. And I have been publishing that to GitHub as I go along. And one of the things I'll be adding to that will be a set of scripts that will go forwards (to augment my scripts going backwards) all the way through the current patch level which is patch-469. And you could go all the way through patch-469 which was released earlier this year and be able to look at things and bring patches forward.

I also hope to create a second set of patches that are just pure patches. If I create a series of patches, then you just download the patches, you run the patch and maybe run a couple of commands to go with it, because even though there's a build everything command in 2.11BSD, when the binary format changes, you have to build the assembler first and install it, and then you have to build the AR program and the loader, and you have to do that in a particular order.

Then you have to build other things, some with old stuff, some with new stuff, so that everything winds up working and you don't break your system and you don't wind up with a system where you can't compile anything to fix it, so you've got to lift something off of tape to restart.

There is a group of people who use what's called a PI DP-11. People have taken historic PDP-11 front panels that you'll find on say a PDP-11/70 and created some glue software so that you can put a raspberry pie in it and interact with it like you could the old PDP-11s. Some members of this community are potentially interested in helping to create the patch chain all the way up to current because—you know—they've had to do this over a span of 30 patches. Since the tape they have is for patch level-430, and they need to go from 430 to 470, that's kind of a pain when you have to apply each one and it's not just patch and rebuild. It's the concept that remains of using scripts or removing this or moving there or building it in this order.

There are a number of people who have created what I'm talking about, going all the way back to the beginning for the last 30 or 40 or 50 patches. One of the hopes is that I can take all the patches and apply the script and actually wind up with the same system that's the snapshot of the other system, or the tape that's there today so as to validate that what I have is a reasonable reconstruction.

Because there are a lot of places in the instructions, it's not as simple as to just run the script and everything will be fine, or here are the five commands you run. They're not at the bottom where the normal commands are, but they're buried in the text. And several of the things that I thought were missing were actually mentioned in the text. Had I read every single line verbatim and remembered it, I would have picked that up, but since there were about 200 patches to go through, I did not do that. That takes a lot of time, and some of these patches are quite verbose.

It has been a kind of a labor of love as well as an interesting series of challenges, and it has tested my knowledge of early binary formats. I wrote a blog that talked about the early binary formats, and that blog explored what the format was on the PDP-11, and it also said, now that you know that, this is what's going on today. And you can take that simplified view and map it onto the more complex view that we have today where we have shared libraries. We have different sections that are treated differently—this is loaded into read-only memory; this is loaded into rewrite memory. And all of these things can be brought in by anything. And so instead of three simple sections, you can have hundreds of more sections in an executing program. A good reason to study history is to understand the current stuff better.

JUDE: Yes, and as you were saying, because the computers were so much more constrained at the time and there was only one CPU and so on, a lot of the code is easier to understand, and looking at that first can help you grasp the concept before you're looking at the more advanced versions in modern operating systems that are much more complicated because there are a lot of moving parts all happening at the same time.

LOSH: Right, understanding an algorithm that is not locked helps you understand how the locking happens in newer incarnations of that algorithm. Oftentimes, when you're first approaching it, the thing you see the most is I have to lock this, do this, unlock that. There is a relationship between the locks, and you can easily get distracted in all of that and not see what's going on underneath. You do not see the big picture because of all the things you have to do to make it perform. It tends to obscure what's going on behind the scenes.

JUDE: As you mentioned earlier with the limited address spaces and so on, sometimes a lot of those tricks maybe still make sense on smaller embedded systems where you only have 16 megabytes of memory to work with. You can't just do what we do nowadays and assume there are gigs and gigs of RAM, it's fine.

LOSH: Exactly.

JUDE: You know, a hundred kilobytes here, a hundred kilobytes there start to add up.

REUSCHLING: Yes, it's all coming back to us. It's a bit like looking at a dinosaur and realizing there are a couple of bones missing. And I try to extrapolate what they would look like from a running mammal that's living nowadays.

LOSH: Or, we have a part of the bone, and we compare that part of the bone and a modern chicken bone or a modern rhinoceros bone because the two animals look similar and say, oh, can we use that to recreate something? This part fits in that so maybe it looked like the modern version.

REUSCHLING: Yes, exactly. So, coming back to the museum part of this question, you mentioned the Unix Heritage Society a couple of times in your blog and your talks, so what is that exactly? And how can people participate in that?

(Part II will appear in the next issue!)

FreeBSD®

The FreeBSD Project is looking for

- Programmers
- Testers
- Researchers
- Tech writers
- Anyone who wants to get involved

Find out more by

Checking out our website
freebsd.org/projects/newbies.html

Downloading the Software
freebsd.org/where.html

We're a welcoming community looking for people like you to help continue developing this robust operating system. Join us!

Already involved?

Don't forget to check out the latest grant opportunities at
freebsdfoundation.org

Help Create the Future. Join the FreeBSD Project!

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system.

Not only is FreeBSD easy to install, but it runs a huge number of applications, offers powerful solutions, and cutting edge features. The best part? It's FREE of charge and comes with full source code.

Did you know that working with a mature, open source project is an excellent way to gain new skills, network with other professionals, and differentiate yourself in a competitive job market? Don't miss this opportunity to work with a diverse and committed community bringing about a better world powered by FreeBSD.

The FreeBSD Community is proudly supported by

FreeBSD FOUNDATION

Building and Running an Open-Source Community—The FreeBSD Project

BY MARSHALL KIRK MCKUSICK

The Berkeley Software Distribution (BSD) started out as a part-time effort of Bill Joy, then a graduate student at UC Berkeley. He ran it out of the office he shared with four other students. Today it has thousands of developers and is used by untold millions of people around the world. This is the story of how we got there.

FreeBSD is an open-source operating system derived from 4.4BSD-Lite released in 1992 and 1994 by the University of California at Berkeley. FreeBSD is one of a very few open-source projects still active after more than 25 years.

The typical life cycle of an open-source project is that a person, or sometimes a small group of people, get excited about doing something and hack up a bunch of code for it. What typically ends up happening is that they finish with the idea, or the leader of the group gets bored and moves on, and the project dies. If you look at SourceForge or GitHub, 90% of those projects are ''dark,'' meaning they have not had any commits to them for at least a year. An open-source project with longevity has to have some method for passing leadership from one person to another. Much of this article looks at FreeBSD's governance structure and how it has evolved over time.

Current Users of FreeBSD

Today, FreeBSD is most heavily used to provide core Internet support. FreeBSD is used to operate root name servers, major web hosts, routing infrastructure, and as the foundation for major commercial operating systems. Some examples include Netflix content distribution servers, which are the source of one third of all Internet traffic; New York Internet, which does much of the network infrastructure on the East Coast; and the BBC, whose servers are primarily running on FreeBSD. Juniper Networks routers and Network Appliance file servers are built on top of FreeBSD. FreeBSD is heavily used in the appliance and embedded operating system market where companies need to put their intellectual property inside the operating system so they cannot use Linux due to its GNU Public License (GPL). I discuss the GPL later in this article.

Verisign uses FreeBSD to handle their critical Internet name service lookup. They use FreeBSD, Linux, and commercial operating systems because they want a diversity of operating systems in case there is a zero-day exploit in one of the operating systems. When a zero-day exploit happens, they can take down the affected systems and not lose their service, as their service must be up and running at all times. They value FreeBSD because it is a small enough niche that there is not a lot of targeting of it. It is hardened enough that it is harder to crack it than to crack other operating systems and part of FreeBSD's security derives from its code base chang-

ing slowly. Less known is that FreeBSD is the operating system used on the Sony PlayStation 4 and later. Apple's Darwin, the base operating system in Mac OS X, is directly derived from FreeBSD.

CSRG at Berkeley

I will start with a bit of history, rolling back the clock to the 1970s and 1980s when BSD was being developed at the University of California at Berkeley. BSD started at Berkeley in 1977 as a one-man project run by Bill Joy (who went on to be one of the founders of Sun Microsystems).

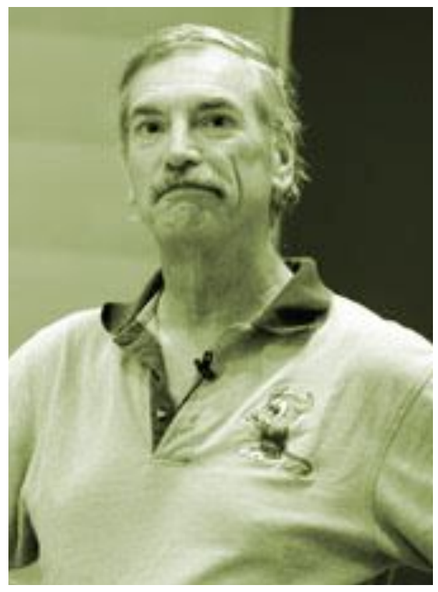


Photo courtesy Ollivier Robert

The first distribution was called the BSD distribution. It was released in early 1977 and consisted of just three utilities that Bill wrote that would run on UNIX:

- the C shell,
- the ex/vi editor,
- the Pascal compiler/interpreter

You would just add these utilities to your existing version of UNIX. Typically, you were running Version 6 UNIX, also known as Sixth Edition UNIX or V6. As users converted to Version 7, which was released in 1979, they could take the BSD utilities with them. There were additional utilities that were added to the 2BSD release in May 1979.

Up to this point, UC Berkeley was running UNIX on Digital Equipment Corporation PDP-11 machines. The PDP-11 had a 16-bit address space and thus had a maximum of 64K bytes of text plus 64K bytes of data. UNIX was a swap-based system. UNIX read programs into memory to run them and freed the memory when they were done.

In 1978, Digital Equipment Corporation released their 32-bit VAX computer (VAX stood for Virtual Address eXtension). Berkeley was an early adopter, receiving a VAX with serial number 7 with its maximum memory capacity of four megabytes. The VAX was the first hardware on which UNIX ran that supported virtual memory. The vendor-supplied operating system for the VAX was VMS, which was mostly a batch system. Its initial front end was a primitive command-line interface for interactive use that was not liked by people who were used to UNIX.

The initial port of UNIX to the VAX (32/V) was done at Bell Labs by John Reiser and Tom London and did not use the virtual memory hardware. It ran as a swap-based system just like the PDP-11. So, you could not run any program larger than physical memory.

Users at UCB were working with Franz Lisp, which needed far more address space than the four megabytes of physical memory on Berkeley's VAX. VMS did support virtual memory, but its interface was not to their liking. So, Bill Joy ported Ozalp Babaoglu's VM system into UNIX /32V over the four-week Christmas break. The port was stable enough by a couple of weeks after the Christmas break, so it permanently replaced VMS usage at Berkeley.

The release containing the VM system became 3BSD and was released in 1979. The 3BSD distribution was the first Berkeley distribution containing the whole kernel, the utilities and libraries, etc. Distributing complete systems became the model of the BSD distributions still used to this day.

The Defense Advanced Research Projects Agency (DARPA) funded a lot of early computer-infrastructure research. DARPA was interested in having a common machine and operating system that would be used across all the DARPA projects so they could easily move code developed by their supported groups between their projects.

DARPA wanted to have other features added, most importantly an implementation of TCP/IP to replace their earlier NCP networking infrastructure as NCP supported only 254 hosts.

Berkeley got one of the DARPA contracts to ''harden'' the BSD kernel and help integrate the networking code. This contract led to the formation of the Berkeley Computer Systems Research Group (CSRG).

After its humble start in Bill Joy's office, the CSRG eventually grew to a staggering four people. They would spend a lot of time talking with other people finding out what they were doing with the system, getting their improvements, their new utilities, and bringing them back and getting them incorporated into the system.

In the 1970s, the network was composed mainly of dialup links running at 1200 baud. Most of the contributions to the CSRG came on a nine-track tape, though occasionally it was sent as an email attachment across a painfully slow link.

In the early 1980s, CRSG started using a source code control system from Bell Labs called SCCS. For the first time, SCCS allowed them to keep track of all the changes that were being made to the system, who made them, and the reasons that they were made. Using a source-code control system was a novel concept back in the early 1980s.

As a DARPA contractor, the CSRG was given access to the ARPAnet (which is what eventually became the Internet). The ARPAnet made it possible to expand the set of people who could work on the system. In particular, the CSRG had network connectivity that allowed others to actually log into their machines remotely, thus avoiding expensive long-distance charges when using modems to connect cross-country or further. Most importantly they had a source code control system so they could avoid collisions and keep track of who was doing what.

Initially, the CSRG gave accounts on CSRG machines to 10 other trusted people who could log on and update sources. This approach was far more efficient than the previous receipt of tapes and email that had to be integrated by CSRG staff. The selected people were those who had been making a lot of contributions and understood the way that CSRG liked to have things done. They would coordinate some set of people with whom they were working, thus increasing the total number of active contributors to about 100 people.

CSRG staff used SCCS to track changes and verify them before doing distributions. A key role of the CSRG was to ensure that the system as a whole remained consistent. Before each distribution, the CSRG team would have the source code control system list everything that had been changed since the last distribution. In those days, the list of changes could be printed out because it was not too huge. The CSRG team would go through and look at every single one of the changes that had been made, make sure that it fit in, and that it didn't break something that was going to affect some other program. When a change was found, say, in the "diff" program that broke the "patch" program, they would contact the folks that made the changes so they could sort out the program that needed to be fixed up.

This structure formed the basis for the current BSD-based projects having:

- a core group overseeing the development,
- a larger set of people who are doing commits into the system, and
- an even larger set of people that are feeding the people doing the commits.

FreeBSD Project Structure

The FreeBSD Project currently uses Subversion for its source-code-control repository. The Project is in the process of moving to using Git as Subversion as Git is more widely used and understood. Anyone can download the repository, but only a subset of the people involved with the project are allowed to update it.

A FreeBSD distribution consists of the core system of about 100 libraries and 750 utilities plus the symmetric-multithreaded kernel that runs on Intel/AMD 32/64-bit, ARM 32/64 bit, MIPS, PowerPC, RISC-V, and other processors.

All other software is maintained in the ports collection that currently has over 32,000 packages. The ports are organized based on different functionality: software development, web

hosting, routing, etc. The ports collection has a database that allows users to look up what they want. The ports come in two flavors: users can either download the source code and build it themselves, or there is a package where it is maintained in a compiled state for the various architectures, so they can just download the binaries.

Releases

FreeBSD has a formal structure for the way that it releases software. The top of the development tree is known as Current and that is where ongoing development is done.

Periodically, FreeBSD branches off from Current and builds a new base distribution referred to as a Stable branch. Bug fixes are first made to Current, then tested. After they have proven to be reasonable and stable, those changes get migrated down into the Stable branches through something called an MFC (move from Current). The checkin to the Stable branch from Current references the particular change that was made to the Current tree. The application binary interface (ABI) is never changed during the lifetime of a Stable branch.

Many projects just have the current release because they do not want to support the older releases as it takes a fair amount of effort to keep maintaining them. Because FreeBSD is used by a lot of companies that build embedded systems, it is important that FreeBSD continue some maintenance of earlier releases for a significant period of time. Historically FreeBSD has attempted to continue providing at least security-level maintenance of the older releases for up to five years.

The FreeBSD Community

Most contributors to FreeBSD are volunteers. A relatively recent phenomenon is that some companies hire people to liaise with the FreeBSD Project and make sure that problems fixed by the company get upstreamed.

The key to understanding an open-source project is to understand how to make the project appealing to volunteers. Volunteers only do what they want to do. They are not working for you as they would have been in a traditional software development company where you have a manager, you are paying your employees, and the manager tells the employees what they are going to do. It may be some task that is not particularly desirable, but somebody will be assigned to do that task, and they will get it done because that is their job.

In an open-source project, the workforce is people volunteering their time. They do not tend to volunteer to do stuff they are not interested in doing. They only do the things they want to do. Their open-source project contributions are their lowest priority (after work, family, recreation, etc.). So, if some crisis occurs at work, or they have family issues, or they want to go on vacation, all those things take precedence over their development. The best of intentions can be preempted by other priorities. The effect of being the lowest priority is that you do not really have a schedule.

Volunteers are transient. Most people who work on an open-source project get excited, they work on it for a while, then they lose interest, and they go off and do other things. Most people do not make a lifetime career working on one particular project. So, projects need to plan for and deal with transience. If they fail to do so, they end up having a lot of deadwood, which eventually weighs the entire project down. Failing to eliminate dead wood is one of the common reasons that projects fail or go dark. The people who are deadwood need to leave.

Open-source projects need to be self-organizing since they typically have no paid staff or managers. To have long-term success, projects need to be democratic and all members should be able to advance up the project hierarchy based on their contributions to the project. Linux is the obvious exception to this rule where Linus Torvolds started and still runs the project. But

even Linux is going to have to find a path to new leadership.

The usual cause of failure for an open-source projects is that the grand leader and their set of friends control the project. A newcomer who wants to get involved can rise up only so far. They can never move into the leadership role, or, often, even become a direct report to the leader. A successful project has to be able to change the leadership. Otherwise the leadership becomes deadwood, which leaves the project rudderless.

A project must anticipate turnover and manage it gracefully. The problem gets back to the whole issue of transients. The project needs to have a system for getting rid of the deadwood. It has to be a very clear and transparent system, where it is obvious that it is not the people in charge who are picking on people. There must be criteria for remaining involved with the project. Everybody has to meet those criteria and when you no longer meet those criteria, it is time to leave. In the FreeBSD Project, the criterion for remaining a project committer is that you must make a commit once every 12 months.

Organization

The organization of the FreeBSD Project is made up of four concentric rings.

Users

The outermost ring is the users. The FreeBSD users can send and receive feedback for bugs and interact on FreeBSD run mailing lists. Most users are not involved in the development, but some will send and receive feedback for bugs.

Figuring out the number of users of an open-source project is difficult. Often the number is based on the count of downloads of the project's software. Projects like to count downloads, then have some multiplicative factor to decide how many users that represents. Since the software is free, many people will download it but never end up using it. At the other extreme, one download might be for an entire company like Netflix that then deploys it across thousands of machines. Figuring out the balance between these extremes is all but impossible. So, the FreeBSD Project just publishes the number of downloads and lets others choose the factor to use.

Developers

Inside the user ring are the developers, who number about 6,000. It is hard to get an exact handle on the number of developers. This estimate is based on how many unique names have posted to bug lists, how many unique names have posted to various mailing lists, and how many people are credited with code coming into the source tree.

The developers are more directly involved in the Project. They are writing code, sending in bug reports, interacting with the Project to get stuff done. Developers have read access to the source-code repository. If they have changes that they want to put back into the system, then they need to find someone who has the right to commit to the source code tree. They then submit their changes to those committers (committers are described in the next subsection). Making a direct connection with a committer is the best and usually quickest way to get a change made though the change can be submitted by sending a pull request.

Committers

Committers are the people permitted to make changes to the system. In recent years, the number of committers has varied between 350 and 400. Most committers are authorized to commit changes to specific parts of the system. Most commonly, the committer will be associated with a particular set of ports in the ports tree for which they agree to assume responsibility. They will have commit privileges for that set of ports. If they want to make changes in other parts of the system, then they need to get their commit privileges expanded.

Ports typically do not have as much oversight as parts of the core system and the kernel. Typically the person in charge of dealing with the port does not need to check with any other committers. The community from which that port comes, for example Apache, already has a lot of oversight going on. So the job of the committer is to keep FreeBSD up-to-date with the current release of Apache. The committer is welcome to go get other opinions, but they are not required to do so, and they generally do not need to do so.

All changes to the core system and the kernel require review by at least one other committer. The commit message must list who did the review. All commit changes are mailed to all committers, so anyone listed as having done a review can speak up if they feel that their feedback had not been resolved.

Committers are the gateway to feed in the changes from the developers. Developers find committers who are working in their area of interest. They typically find the relevant committer by looking at who has been actively making changes in the code that they are interested in changing and contacting them directly.

They work together to get their changes put in. When one of these developers begins to become more involved, the committer can nominate that developer to become a committer. The process of becoming a committer is somewhat involved. Once a developer gets nominated by a committer, the Core Team (described in the next section) will make a decision.

A new committer is required to have a mentor who is typically the committer that nominated them. The mentor is responsible for ensuring that the committer is not shooting themselves in the foot. The mentor helps the developer learn what they need to know: how the source-code control workflow is managed; how to use Phabricator (the forum that provides pre-commit code review workflows); how to use Bugzilla (the forum that tracks bug reports); where to find out the Project's coding styles; on which mailing lists to post review requests; which changes require a broader review; even small details such as on which IRC channel they should engage.

Mentorship generally ranges from a few months to more than a year. Somebody who is very active in the Project can get through the mentoring process much more quickly than someone who is working more intermittently. Most of the committers appreciate having a mentor as they do not want to do something that is going to be viewed as wildly stupid or out of track with the way the Project works. Often even after the mentor releases them to be on their own, they still continue to request the mentor to keep reviewing their changes.

The Project needs to deal with the unfortunate but important task of identifying and retiring commit privileges from the committers who are no longer contributing. The Project needs to have a well-defined and uniformly applied policy in place to retire inactive committers.

The FreeBSD retirement policy is the automatic suspension of commit privileges after one year of non-use. For the six months after suspension, commit privileges can be restored by simply requesting that they be reactivated. After 18 months of non-use and six months of being suspended, a retired committer has to go through the whole nomination and mentoring process again.

The reason for repeating the nomination and mentoring process is that the way the Project works changes over time. New rules and procedures get put into place. Major tools may have changed, such as a switch in the source code control system. New tools such as the Phabricator review system may have been added. There may have been changes in the bug-tracking system. Often there has been a reorganization of the mailing lists, IRC channels, etc. The returning committer needs somebody to get them back up to speed. The mentoring period is often short for people returning to the Project as committers.

Avoiding the loss of commit privileges is really a minor threshold to get over. Retaining commit privileges requires only one commit per year and can be done by updating your personal

information (what people can see when they pull up ''who is this person''). So, the hurdle is about a millimeter high.

A recent study of the Project demographics found that the average age of the committers is 39; the median age is 37; the youngest is 25; the oldest is 68; and the biggest group is concentrated in the 31 to 40 age range.

Most of the people coming onto the Project have been doing software development for at least a decade. They typically start in their teens so by their mid-to-late 20s they have 10 years under their belts. Many of the people come from the Linux world where they have risen up as far as they can go. They are feeling as if they would like to be able to move up further. They find out about FreeBSD, they see how it is set up and they like it, so they come over and join the Project.

While the Project has had committers in their teens, the benefit of a slightly older demographic is that they are past their "youth angst," which means that they have gotten over squabbling on mailing lists. They tend to be a little more mature in their discussions with far fewer ad hominem attacks.

It is important to avoid fights on the mailing list, particularly ad hominem attacks. So the mailing lists are closely monitored. If people start getting into too much of a rat hole or just personal attacks, they will be pulled aside and told to take that off the list. It does not take much to poison a list and it is very hard to get it back on track once that has happened.

Core

The Project is headed up by the nine-member Core Team. Their role is to oversee and manage the FreeBSD Project.

The history of who is in charge of the Project evolved over time. When the BSD Project started at Berkeley it used ''the lord and master'' approach. A single person made the final decisions. The BSD Project was started and run by Bill Joy until he left to go to Sun Microsystems. Mike Karels took over and then later Kirk McKusick (your humble narrator and author of this article).

When the BSD Project was spun off by Berkeley and it passed to the outside world, several distributions sprang up, including those by Bill Jolitz and later NetBSD and FreeBSD. This article focuses on the evolution of FreeBSD.

When the FreeBSD organization was set up, the organizers decided to set up a group of seven people called the Core group that were in charge of overseeing the Project. The original Core group was self-selected. The people who set up the Project deputized themselves onto the Core Team. They were ''Tsars for life.''

As is common in open-source projects, some began to lose interest. There was no mechanism to replace them and most did not want to leave because of the prestige of being on the Core Team.

A group of people got together and decided to write a set of bylaws that included a provision to make Core an elected position. Core was also expanded to nine people. The entire Core is elected every two years. Core members are nominated from and elected by the committers. Any active committer can run for Core. Candidates are self-selecting and no nomination is required.

Each candidate puts up a statement as to why they should be elected to Core. In recent years, there have been a series of "office hours" where Core nominees show up to answer questions and expand on their vision for the Project.

After a prescribed number of weeks, the election is held. Every committer gets nine votes. The top nine vote-getters become the next Core. Though there have been efforts to install term limits, at present there are none. The electorate are very cognizant of who on Core is get-

ting things done and who works and plays well with others. The deadwood can continue to run, but they tend not to get reelected. Historically three to four Core members have turned over every two years.

A primary responsibility of the Core Team is to oversee the groups that run the Project. These include:

- the group that administers the Project servers,
- the group that does release engineering,
- the group that manages the QA and continuous integration testing,
- the people that organize and run FreeBSD summits,
- the group that handles security issues,
- the group that oversees the ports repository,
- and the group that oversees the system documentation.

An important task is to organize running the FreeBSD Summits where people actually using the system talk about what they are doing, what they need, and what they would like to see. A key takeaway from a summit is a list of tasks for which someone has volunteered to do something. Indeed, the summits often provide a wealth of ideas that become part of the roadmap that lays out the big vision of what needs to be done. Most important is getting the roadmap implemented, which involves encouraging people toward getting some of their ideas implemented.

Core is also responsible for adding and removing committers. As already described, requests for bringing in a new committer are submitted to Core. Core reviews the request, ensures that one or more suitable mentors have been found, and if all is in order approves the request. Committer removal typically occurs through the timeout mechanism described above.

Another important role for the Core Team is to resolve differences between committers. Most of the time, differences get hashed out on a mailing list or an IRC channel. Sometimes there is a difference in philosophy, and the parties are unable to sort it out. Occasionally, these devolve into “commit wars.” Something is checked in, it gets ripped out, it is checked back in again, etc. Here, Core needs to step in and arbitrate.

The tool that Core has to enforce decisions is the ability to temporarily suspend commit privileges. For example, in resolving a commit war, Core can impose a cooling-down period. Neither party will be allowed to commit anything for two weeks. Very rarely, a committer’s commit privileges can be permanently removed. Complete removal has only happened twice in the history of the Project. A very small number considering the many thousands of committers over the history of the Project.

Types of Contributors

There are many different types of contributions and it is important to have a broad view. There is not a hierarchy where kernel committers are better than utilities committers, are better than documentation committers, etc. Some other projects have hierarchies such as these but in the FreeBSD Project there is no higher value attributed to one type of contribution over another. People contribute in the areas in which they are skilled.

Some of the areas in which people contribute:

- Port maintainers (196 committers made 27,840 commits to Current in last 12 months). These commits do not count MFCs and other changes that would increase the total by perhaps 50%.
- Utility maintainers (193 committers made 4,334 commits to Current in last 12 months). They are maintaining 775 utilities and libraries. These commits do not count the one to two

additional MFCs for each of these commits.

- Kernel maintainers (71 committers made 6,056 commits to Current in last 12 months). These commits do not count the one to two additional MFCs for each of these commits.
- Documentation group (75 committers made 2,076 commits to Current in last 12 months). One commit is likely an entire section of a manual, or an entire manual (man) page, which are typically much larger than the coding commits. One of the strengths of the FreeBSD Project is that it has better documentation than most other open-source projects. Today the documentation is set up with multi-language support. Most documentation is available in about 10 languages.
- Security team (7 members). Most of the time, they do not have a lot to do, but when a security threat comes up, they have to jump on it now! For example, Heartbleed. Having seven team members allows coverage 24 hours per day. There are people in the U.S., Australia, Japan, India, Eastern Europe, and Western Europe.
- System administrators (7 members composed of 5 website and 2 email). A thankless but very necessary task. The Project has websites, a lot of online documentation, numerous mirrors that need to be maintained, mailing lists, build servers, development machines, etc.
- Release engineers (1 main, 10 helpers) During the actual release phase, the number of helpers jumps dramatically.
- Quality assurance (2 main, 5 helpers). This is the newest group at FreeBSD. Many open-source projects do not have such a group.

Advocacy and marketing group (2 main, 6 helpers). Around a decade ago, a member of this team was elected to Core. It led to a logo change and to a total website overhaul. These changes demonstrate that Core is about more than coding; it is also about running and promoting the Project.

Licensing

Most commercial software is developed using a traditional copyright. Usually the source is not available or is only available in very restricted ways often requiring the signing of a nondisclosure agreement. Recipients may not even be allowed to modify the source and certainly cannot pass it out to anybody else. Since open-source projects generally want their source code to be given away, they do not use traditional copyrights.

Open-source software commonly uses the GNU Public License (GPL), sometimes referred to as a ''Copyleft'' license. Software covered by the GPL version 2 (GPL 2) license must make source code available including any changes that have been made to it. With the more restrictive GPL version 3 (GPL 3) license, in addition to making your source code available, if any of that source code is covered by your patents, you have to make those patents available at no cost for use by anyone who uses your code. The GPL 3 license has caused a lot of consternation. Many companies got around the GPL 2 license by patenting the ideas in their code. They release their source code as required by the GPL 2 license but then require anyone using it to pay patent royalties. This approach was against the spirit of GPL which is why the GPL 3 license was created.

The Linux kernel is under a GPL 2 license, which means that any code written to run as part of the Linux kernel must be released. Some companies use their patents to control its use by others. Many companies avoid having to release their source code by creating loadable binary modules that are loaded into a Linux kernel. Loadable modules are a controversial way of avoiding the GPL terms.

Linux has not changed over to a GPL 3 license. But a lot of the other GNU software is under the GPL 3 license, including most of the rest of the software that is packaged around the Linux

kernel including the GCC compiler and its libraries. It is not yet clear if, when loading the GPL 3 libraries, your program becomes subject to the GPL. The GPL 3 license has made a lot of companies nervous, which has helped projects with a Berkeley license.

The other common approach is the Berkeley license. Sometimes referred to as a ''Copy-center'' license, as in take it down to the copy center and make as many copies as you want. Source and patent rights may or may not be provided, i.e., you can give back your changes, or not, as you choose.

FreeBSD uses a Berkeley license, which has played a big role in its success, particularly with companies that have their proprietary code in the kernel. In practice, the FreeBSD Project gets back about as much code as do GPL open-source projects. But it takes longer as there is a learning curve.

Company X builds their proprietary product on FreeBSD and they choose not to give anything back. Later, the time comes to upgrade the underlying FreeBSD to the next version. They need to port many bug fixes they have made, and they have to port all of their changes. It is a long and expensive process. They realize that if they had given back their bug fixes they would have just been there. So, this experience leads to an incremental amount of bug fixes coming back. Two years hence, they upgrade to the next release. They realize that many of their changes are not really proprietary. If they contribute them back, then someone else will maintain them. By the third upgrade cycle, the company is trying to give back code that is specific to their product. The result is that companies start hiring committers to give them a direct channel to what is changing in the system and to facilitate getting their bug fixes and enhancements merged back into FreeBSD.

Conclusions

Building an open-source community requires an ongoing effort. It is tempting to cling to doing things the way that they have always been done. To maintain its vitality, the Project must adapt to the needs of its users as they evolve over time. The community must be welcoming to newcomers and provide a path for their advancement. Newcomers need mentors to help them get integrated into the community. At the same time, the mentors need to accept the new approaches that are being brought to the Project. The FreeBSD Project has managed to successfully balance these needs for more than a quarter of a century. I am hopeful that they will continue to do so for many more years to come.

DR. MARSHALL KIRK MCKUSICK writes books and articles, teaches classes on UNIX- and BSD-related subjects, and provides expert-witness testimony on software patent, trade secret, and copyright issues. He has been a developer and committer to the FreeBSD Project since its founding in 1993. While at the University of California at Berkeley, he implemented the 4.2BSD fast filesystem and was the Research Computer Scientist at Berkeley overseeing the development and release of 4.3BSD and 4.4BSD.

FreeBSD® JOURNAL

The FreeBSD Journal is Now Free!

Yep, that's right Free.

The voice of the FreeBSD Community and the BEST way to keep up with the latest releases and new developments in FreeBSD is now openly available to everyone.

DON'T MISS A SINGLE ISSUE!

2020 Editorial Calendar

- FreeBSD in Research (Jan/Feb 2020)
- Filesystems (March/April 2020)
- Network Performance (May/June 2020)
- Benchmarking/Tuning (July/August 2020)
- Contributing/Onboarding (Sept/Oct 2020)
- Workflows (Nov/Dec 2020)

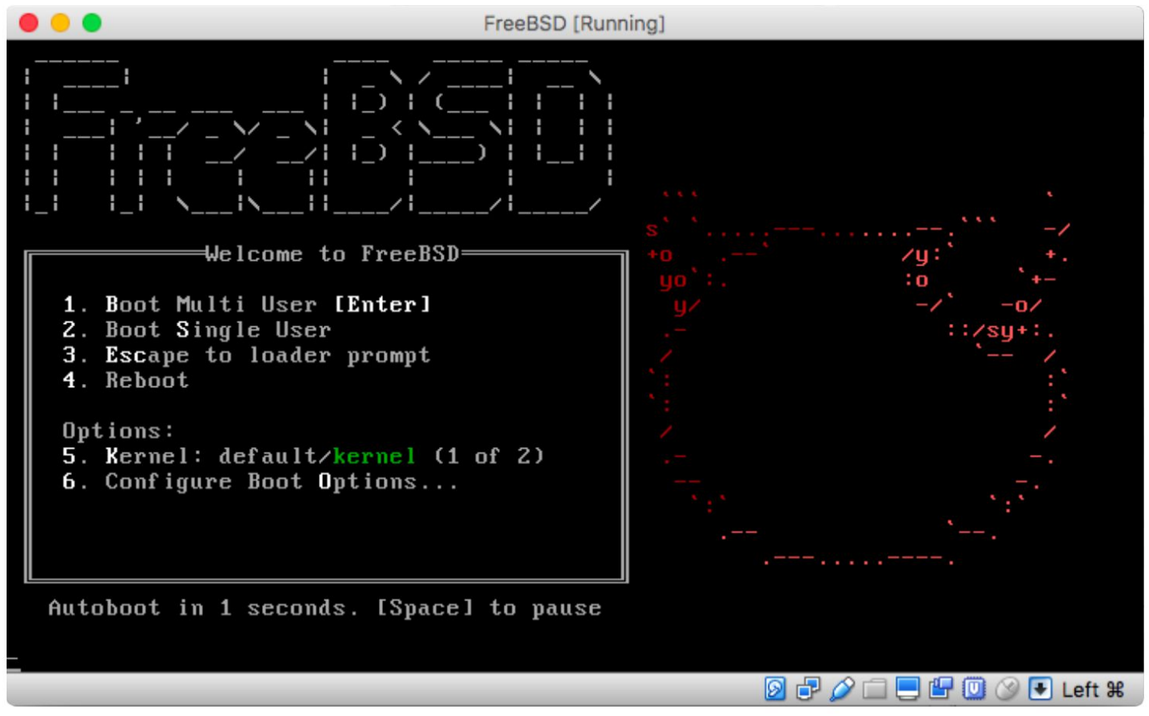
Find out more at: freebsdfoundation/journal

FreeBSD® guides

Easy Minecraft Server on FreeBSD

BY DREW GURKOWSKI

Step 1: Choosing the Device



Before setting up the server, choose between setting up a dedicated device or a virtual machine running FreeBSD.

- A dedicated server will be much more dependable but requires a device completely dedicated to running the server. This option is great if looking for a server with full-time dependability.
- Running the server on a virtual machine will allow the user to run a server without a dedicated device as well as giving the user more control over when the server is running.

If the device or virtual machine is already running FreeBSD, continue to the next step. Otherwise, consider one of the Foundation's guides to get set up with FreeBSD.

- Installing FreeBSD with VirtualBox
- FreeBSD as a Primary Operating System
- Installing FreeBSD for Raspberry Pi

Step 2: Using the Ports Collection

FreeBSD Ports

All commands in this section should be run as root; using the "`su`" command will also work. To download a compressed snapshot of the Ports Collection into `/var/db/portsnap`:

```
# portsnap fetch
```

When running Portsnap for the first time, extract the snapshot into `/usr/ports`:

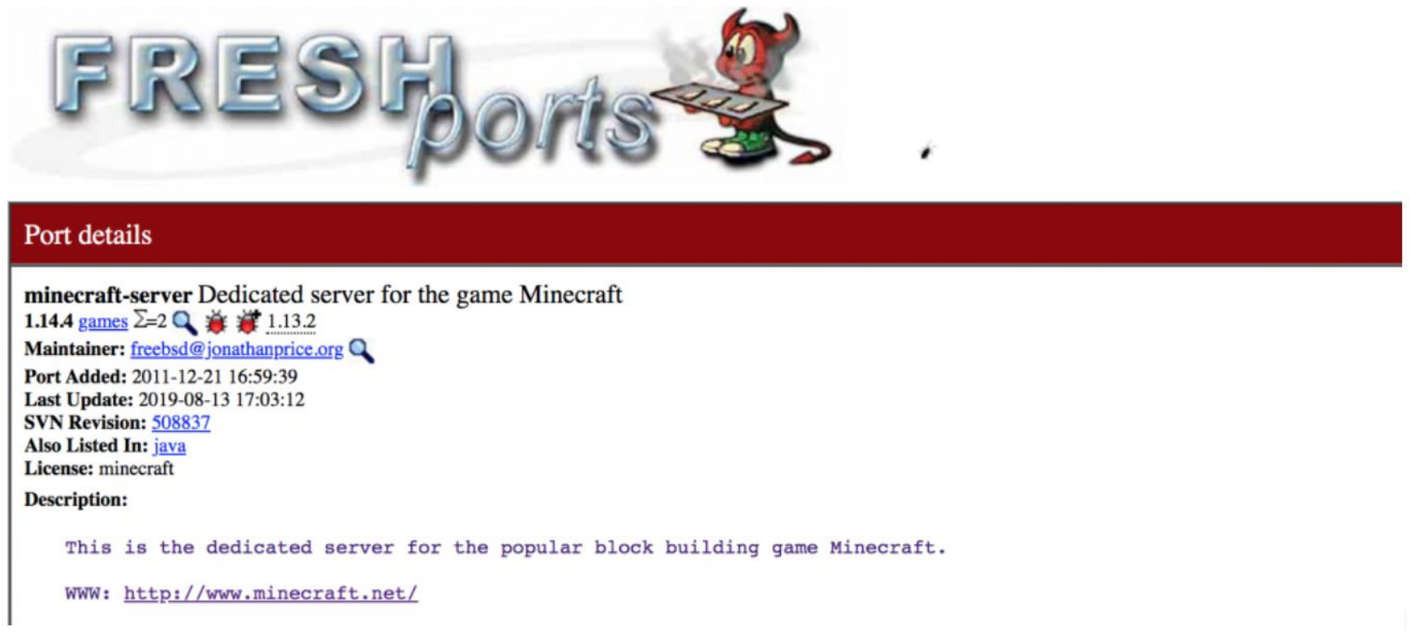
```
# portsnap extract
```

After the first use of Portsnap has been completed, as shown above, `/usr/ports` can be updated as needed by running:

```
# portsnap fetch update
```

Step 3: The minecraft-server Port

The FreeBSD ports collection has a wide variety of gaming ports available, including a great option for running a basic Minecraft server: the minecraft-server port.



The first step will be to install the port; now that the ports collection has been added, this can be done simply with the command:

```
# cd /usr/ports/games/minecraft-server/ && make install clean
```

To populate the necessary files, the user will need to execute the program with the command:

```
# minecraft-server
```

Next, the user will need to agree to the Minecraft End User License Agreement (EULA). This can be done by using the vi command. The following command will bring up the vi text editor:

```
# vi /usr/local/etc/minecraft-server/eula.txt
```

Using the editor (use the manual page if unfamiliar with the editor) change the 4th line to read:

```
eula=true
```

The server can then be run with the command:

```
# minecraft-server
```

To stop the server:

```
# stop
```

Step 4: Identify The External IP Address

To connect to the Minecraft server over the Internet, the user will need the external IP of the device as well as to configure the router's port forwarding. This process will be different for every single type of router, so this guide provides a few links to make this easier.

First, identify the external IP of the device. A super easy way of doing this is using the `wget(1)` command:

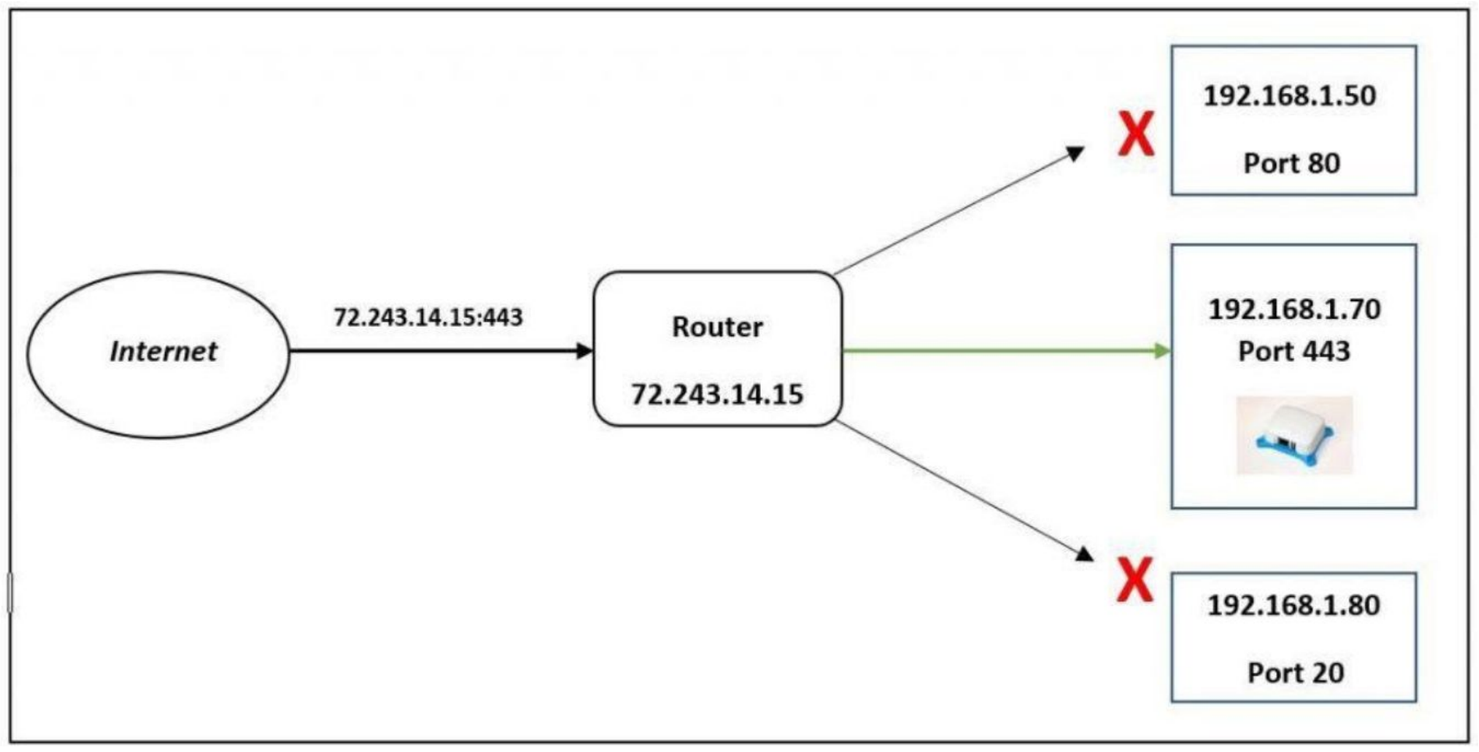
```
# pkg install wget
```

proceed through the installation, typing 'y' when prompted. Next, use `wget(1)` to pull the IP address from a website that will identify the device's external IP.

```
# wget -qO - http://wtfismyip.com/text
```

This should return an IP that looks similar to "65.214.224.57"; however, each set of numbers will be different. This is the external IP address. Other devices will need this string to connect to the Minecraft server.

Step 5: Port Forwarding



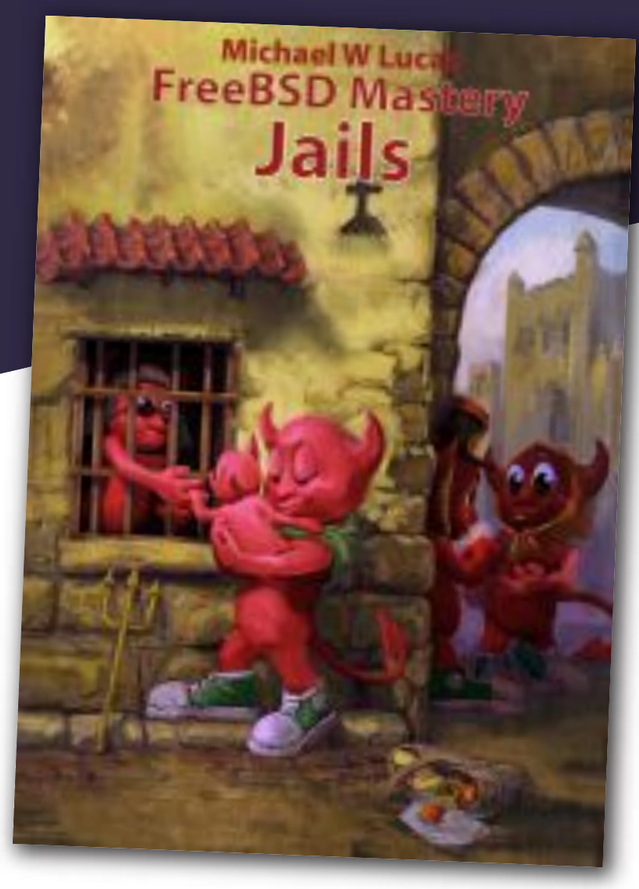
The next step is to make sure that the router connected to the device is properly forwarding all Minecraft traffic to the device. Even if there are no other computers on the network, this is a

Jails ARE FBSD'S MOST LEGENDARY FEATURE:

KNOWN TO BE POWERFUL, TRICKY TO MASTER, AND CLOAKED IN DECADES OF DUBIOUS LORE.

FreeBSD Mastery: Jails cuts through the clutter to expose the inner mechanisms of jails and unleash their power in your service.

Confine Your Software!



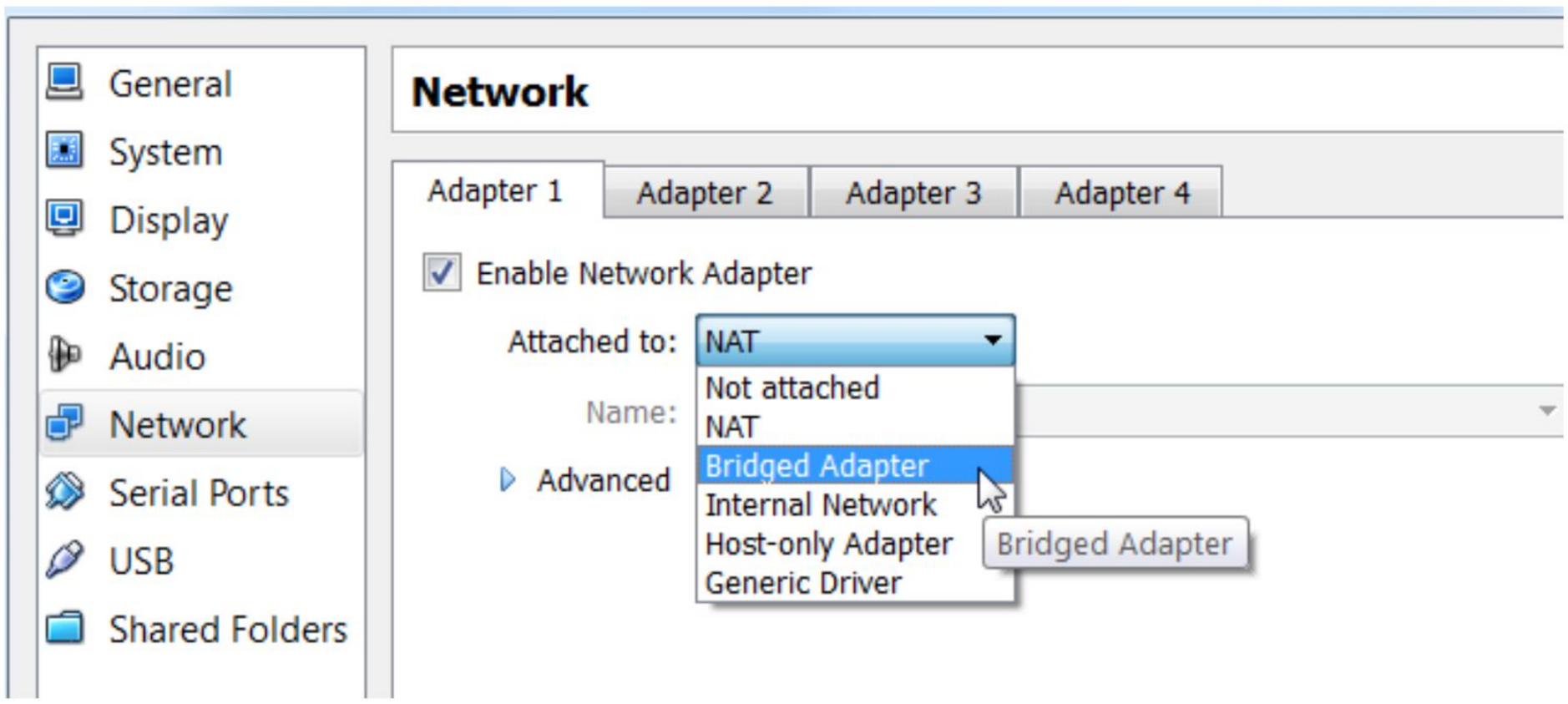
* Understand how jails achieve lightweight virtualization
* Understand the base system's jail tools and the iocage toolkit
* Optimally configure hardware
* Manage jails from the host and from within the jail
* Optimize disk space usage to support thousands of jails
* Comfortably work within the limits of jails
* Implement fine-grained control of jail features
* Build virtual networks
* Deploy hierarchical jails
* Constrain jail resource usage.

...And much, much more!

FreeBSD Mastery: Jails BY MICHAEL W LUCAS ***Available at All Bookstores***

required step. Minecraft uses port 25565, so the router needs to be set up to send any traffic on port 25565 to the computer that has the Minecraft server. Router setup will have to be done in a separate computer with a desktop environment set up.

Only do this step if on a virtual machine: Many virtual machines will have the same IP address as the host computer but will receive none of the incoming traffic. To remedy this, a bridged network adapter will need to be used. This can be changed by opening the virtual machine's network settings and switching the network adapter from NAT to bridged. This will give the virtual machine its own unique IP address that can be used to forward traffic. It is important to use this and not the host computer's IP address in the next few steps, as traffic will not be correctly forwarded to the Minecraft server.



During setup, the router will have to identify the correct device to forward traffic to, while many routers will list the devices and the internal IP (different than the one found in step 4), this may not be the case. This IP should resemble 192.168.1.176 and is the number assigned to the device by the router. The command to identify this IP on the device is:

```
$ ifconfig
```

Each router will have a unique set-up process for port forwarding. Visit portforward.com and select the router model (usually listed on the back of the router). If an ad appears, click CLOSE, the application being advertised is not needed.

The next page will have a guide to set-up port forwarding on that model of router, while each router is different, the important ports to forward to the device are listed below:

TCP: 25565
UDP: 19132-19133,25565

The external and local IP addresses may change when the device is shut down or the modem is reset. Each time the server is restarted, check that these have not changed and update the settings as required.

Step 6: Connecting to the Server

Now that the server has been set up, any Minecraft client should be able to connect with the external IP address found in step 4. First, ensure that the server is running on the host device or VM with the command:

```
# minecraft-server
```

Next, boot up the Minecraft client on the computer that will be connecting to the server, click **Multiplayer** and then **Direct Connect**; the client will then ask for the server address. This is the external IP found on step 4 of this guide. Clicking **Join Server** will then start the game session.

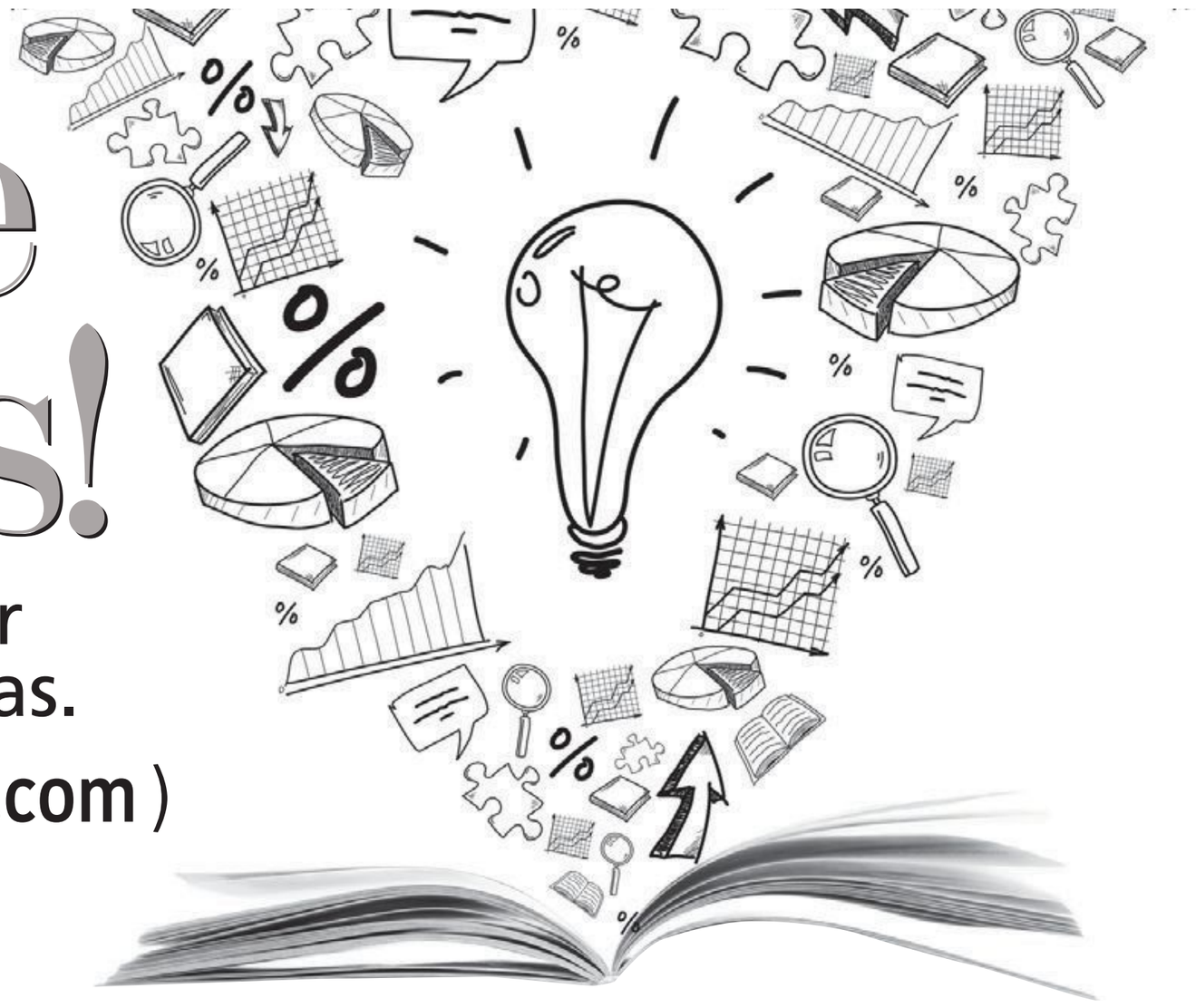
Once you have started playing on the server you can play around with server commands; these will be entered in the terminal on the device running the server and can be used to manipulate the game world, game rules, and players allowed on the server.

DREW GURKOWSKI, FreeBSD Foundation

Write For Us!

Contact Jim Maurer with your article ideas.

(jmaurer@freebsdjournal.com)



Support FreeBSD®

Donate to the Foundation!

You already know that FreeBSD is an internationally recognized leader in providing a high-performance, secure, and stable operating system. It's because of you. Your donations have a direct impact on the Project.

Please consider making a gift to support FreeBSD for the coming year. It's only with your help that we can continue and increase our support to make FreeBSD the high-performance, secure, and reliable OS you know and love!

Your investment will help:

- Funding Projects to Advance FreeBSD
- Increasing Our FreeBSD Advocacy and Marketing Efforts
- Providing Additional Conference Resources and Travel Grants
- Continued Development of the FreeBSD Journal
- Protecting FreeBSD IP and Providing Legal Support to the Project
- Purchasing Hardware to Build and Improve FreeBSD Project Infrastructure

Making a donation is quick and easy.

freebsdfoundation.org/donate

FreeBSD™
FOUNDATION

2020 Events Calendar

BSD Events taking place through October 2020

BY ANNE DICKISON

FreeBSD Fridays: A Series of 101 Classes

Take place every other Friday, beginning July 17, 10 a.m. PDT/17:00 UTC

VIRTUAL

Please continue to join us for our expanded series of virtual 101 classes. We're continuing to build our FreeBSD Fridays content to offer one-hour live sessions designed to get you started with FreeBSD. Our past classes, still available for viewing, are listed at the end.

September 25: Introduction to Jails — Michael W Lucas
Jails are widely considered one of FreeBSD's killer features, but they're widely misunderstood. This talk takes you through how jails work, what they can and can't do, and how to decide how jails fit into your environment.

We'll discuss:

- Jails as VMs
- Configuring the jail host
- Properties and parameters
- Jail management
- Packages and upgrades
- Base jails
- Virtual networking with VNET
- Firewalls in jails
- Jails in jails
- Resource restrictions

Past classes include:

July 17: Introduction to FreeBSD — Deb Goodkin
Foundation Executive Director walks you through the history of FreeBSD, how the FreeBSD Project works, and how to get started as a contributor.

July 24: FreeBSD InstallFest Part 1 — Roller Angel
In part one of the FreeBSD InstallFest, Roller Angel takes you through the process of downloading, installing, and configuring your FreeBSD instance on a virtual machine.

July 31: FreeBSD InstallFest Part 2 — Roller Angel
In part two of the FreeBSD InstallFest, Roller Angel takes you through the process of setting up the desktop and connecting to the Internet with a web browser.

August 7: History of BSD Fast Filesystem — Dr. Marshall Kirk McKusick
This FreeBSD Friday installment is a little different. We're working in collaboration with the USENIX Association and Dr. Marshall Kirk McKusick to bring you the History of the BSD Fast Filesystem. Kirk begins the session with a brief introduction, followed by the USENIX FAST '15 keynote: A Brief History of the BSD Filesystem. Following the presentation, Kirk is back for a live Q&A.

Events Calendar

August 14: Introduction to ZFS — Dan Langille

Dan Langille takes you through the basics of ZFS including:

- A short history of the origins
- An overview of how ZFS works
- Replacing a failed drive
- Why you don't want a RAID card
- Scalability
- Data integrity (detection of file corruption)
- Why you'll love snapshots
- Sending of filesystems to remote servers
- Creating a mirror
- How to create a ZFS array with multiple drives which can lose up to 3 drives without loss of data
- Mounting datasets anywhere in other datasets
- Using ZFS to save your current install before upgrading it
- Simple recommendations for ZFS arrays
- Why single drive ZFS is better than no ZFS
- No, you don't need ECC
- Quotas
- Monitoring ZFS

August 28: Introduction to FreeBSD Security — Antranig Vartanian

FreeBSD is known for its stability and security. Many people know about the firewalls that come with it; however, other subsystems like audit and MAC can give value both long-term and short-term as well! This talk takes you through the IPFW firewall for workstations and server, Security Event Auditing, and where to go beyond that.

We discuss:

- Firewalls in FreeBSD
- IPFW for workstations
- IPFW custom rules
- Security Event Auditing
- BSM Setup
- What's else in FreeBSD Security?

September 11: Introduction to Hardware Hacking with Raspberry Pi — Tom Jones

This is an introduction to FreeBSD running on the Raspberry Pi and will cover interfacing with hardware like LEDs and switches.

FreeBSD Office Hours

https://wiki.freebsd.org/OfficeHours

Next scheduled is September 19, 2020, 7 a.m. to 2 p.m. PDT/14:00 to 21:00 UTC

Topic: Bug squash. Hackathon focused on closing PRs.

Facilitator(s): Tom Jones and Allan Jude

Join members of the FreeBSD community for FreeBSD Office Hours. From general Q&A to topic-based demos and tutorials, Office Hours is a great way to get answers to your FreeBSD-related questions. Past episodes can be found at the FreeBSD YouTube Channel (https://www.youtube.com/c/FreeBSDProject).

Events Calendar

OpenZFS Developer Summit 2020

October 6-7, 2020

Virtual

The eighth annual OpenZFS Developer Summit, which will be a virtual event, held October 6-7, 2020. As with previous years: The goal of the event is to foster cross-community discussions of OpenZFS work and to make progress on some of the projects that have been proposed. This 2-day event consists of a day of presentation and a 1-day hackathon. The Foundation is pleased to be a Bronze Sponsor.

nerdear.la 2020

October 20-24, 2020

Virtual

Building on the success of 2019. nerdear. la will be an online event dedicated to the popularization and advancement of science and technology. Members of the Foundation team will present a brief history of the Project and where it is today, and what the future might hold.

Write For Us!

Contact Jim Maurer
with your article ideas.

(jmaurer@freebsdjournal.com)